UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

André Luís Duarte

Junção Canalizada

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para a obtenção do Título de Mestre em Ciência e Tecnologia da Computação.

Área de Concentração: Sistema de Computação.

Orientador: Prof. Dr. Enzo Seraphim

Abril de 2012
Itajubá – MG

Ficha catalográfica elaborada pela Biblioteca Mauá –
Bibliotecária Cristiane N. C. Carpinteiro- CRB_6/1702

D812j

Duarte, André Luis
Junção canalizada. / por André Luis Duarte. -- Itajubá (MG) : [s.n.], 2012.

57 p.: il.

Orientador: Prof. Dr.Enzo Seraphim.
Dissertação (Mestrado) – Universidade Federal de Itajubá.

1. Operador de junção. 2. Algoritmo. 3. Banco de dados. 4. Dados métricos. I. Seraphim, Enzo, orient. II. Universidade Federal de Itajubá. III. Título.

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Junção Canalizada [1]

[1]Este trabalho conta com o apoio financeiro da CAPES

Em memória de minha mãe Gessy e meu pai Haroldo! Sinto saudades...

"Até aqui, nos ajudou o Senhor."
I Samuel 7:12

Agradecimentos

Agradeço a Deus por ter dado Seu único Filho para morrer por mim na cruz e permitir que hoje eu esteja realizando meus sonhos, certo de que, um dia, estaremos juntos. Agradeço à minha amada esposa Jussara e aos meus filhos Lucas ("Magrelo"), Davi ("Amigão") e Samuel ("Piquinininho") por serem o que mais tenho de precioso neste mundo. Agradeço ao meu grande amigo e orientador professor Dr. Enzo Seraphim por ter feito por mim mais do que orientar, por ter sido um exemplo de profissional e de ser humano: obrigado por ter compartilhado comigo momentos que só um irmão poderia proporcionar. Agradeço ao professor Dr. Edmilson Marmo Moreira pelas conversas que me davam ânimo, pois nelas eu via um grande exemplo de homem, professor e amigo. Agradeço também ao professor Dr. Benedito Isaías de Lima Lopes por ter acreditado em mim no dia da entrevista e pelo abraço que me deu em um dos dias mais tristes da minha vida. Agradeço aos demais professores que me deram, gratuitamente, seus mais preciosos conhecimentos, que Deus recompense abundantemente a todos. Agradeço à minha igreja e à minha família que intercederam e torceram por mim. Agradeço aos professores Dr. Roberto Affonso da Costa Júnior e Dr. Roberto Claudino da Silva que me atenderam e me ajudaram quando precisei, ao professor Carlos Minoru Tamaki pela atenção quando fui seu colega de trabalho. Agradeço aos meus amigos e colegas, em especial ao Cláudio, ao Maurício e ao João pelos momentos de descontração e pela ajuda na obtenção desta vitória. Agradeço à Cristina Silva (Cris) pela paciência em nos atender e sua garra para nos ajudar em tudo, sempre nos recebendo com um belo sorriso na PRPPG. Agradeço à CAPES por ter me dado o apoio financeiro durante o mestrado. Por fim agradeço à Juliana Campos pela revisão desta dissertação. Que Deus abençoe e retribua ricamente a todos.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Dentre os diversos operadores de consulta em banco de dados, a junção é uma operação binária que permite representar dinamicamente os relacionamentos existentes entre as diversas relações em um sistema de gerenciamento de banco de dados.

Pela importância e complexidade, a operação de junção é alvo de muitos estudos e discussões, visto que ela, geralmente, consome muito tempo de processamento e demanda uma quantidade maior de acessos a disco do que outras operações em banco de dados. Por conta disso, propor implementação de junção significa preocupar-se com leituras e escritas realizadas pelos acessos em dispositivo secundário (disco) e com operações de processamento realizadas pelos cálculos e comparações entre as tuplas das relações envolvidas.

Assim, este trabalho tem o objetivo de explorar os algoritmos de junção aplicados a conjuntos de dados métricos que estão sujeitos a geometria euclidiana e esférica. Os conjuntos de dados envolvidos são pontos e rotas armazenados, respectivamente, em uma estrutura métrica e um grafo acíclico de grau máximo dois que formam um caminho. Como resultado, foi proposto um novo operador de junção que responde consultas do tipo: "*Quais são os supermercados (pontos indexados) que estão até a distância de 1 km da rota do caminho de casa ao escritório (grafo acíclico de grau máximo 2)*".

Abstract

Among the various query operators in a database the join is a binary operation that allows represent dynamically the relationships between the various relations in a database management system.

Given the importance and complexity the join operation is the target of many studies and discussions since it usually consumes much processing time and demand a larger amount of disk accesses than other operations in the database. Because of this proposed implementation of joint hits on secondary device (disk) and processing operations carried out by the calculations and comparisons between the tuples of the relations involved.

Therefore this study aims to explore the join algorithms applied to a set of metric data which are subject to Euclidean Geometry and Spherical Geometry. The datasets involved are points and routes stored respectively in a metric structure and acyclic graph of maximum degree two that form a path. As a result we proposed a new join operator that answers queries like: "*What are the supermarkets (indexed points) that are to a distance of 1 km of the route that represents the way home to the office (acyclic graph of maximum degree 2).*"

Capítulo 1
Introdução

Durante as décadas de 60 e 70 surgiram os primeiros esforços da comunidade de pesquisa no sentido de criar uma forma de manipular de maneira eficiente e segura os dados produzidos pelos diversos segmentos da sociedade. Esta manipulação ocorre nos domínios de dados numéricos e pequenas sequências de caracteres. Esses conjuntos possuem relação de ordem total entre os seus elementos, ou seja, dado um conjunto *S*, as relações entre seus elementos obedecem às seguintes propriedades: *totalidade*: $\forall a, b \in S, (a \leq b \vee b \leq a)$, *anti-simetria*: $\forall a, b \in S, (a \leq b \wedge b \leq a \Rightarrow a = b)$ e *transitividade*: $\forall a, b, c \in S, (a \leq b \wedge b \leq c \Rightarrow a \leq c)$. Dessa forma, os operadores relacionais $>$, $<$, $\leq$, $\geq$, $=$ e $\neq$ são usados para a construção de uma grande quantidade de algoritmos e estruturas de dados que manipulam de maneira eficiente os elementos destes conjuntos.

Para a manipulação de conjuntos de dados em domínios que possuem relação de ordem total, estruturas como os vetores tinham um grande destaque pela facilidade na sua implementação e manipulação. À medida em que os conjuntos ficavam grandes, percebeu-se que operações simples passaram a demandar muito tempo de processamento, as inserções se tornavam lentas, pois os elementos do vetor tinham que ser deslocados para manter a relação de ordem total entre eles. Assim, quanto maior o vetor, mais tempo era gasto para o seu processamento. Frente a este problema, pesquisas permitiram o surgimento de novas formas de organizar estes conjuntos de modo mais eficiente dando lugar, assim, às estruturas chamadas árvores. Com as árvores foi possível indexar os elementos de um conjunto de forma mais eficiente do que em vetores, fazendo com que este tipo de estrutura se tornasse o centro das pesquisas em diversas áreas da computação.

Neste cenário, os pesquisadores construíram estruturas de dados e algoritmos que as manipulavam cada vez mais rápido. Consultas e atualizações de elementos se beneficiavam da relação de ordem total inerente aos dados envolvidos. Ainda hoje muitos destes algoritmos e estruturas são usados em banco de dados como, por exemplo, a *B-Tree* [5].

A partir da década de 80, com a utilização de novas formas de captura de dados e o crescimento da

capacidade de armazenamento, surgiram outros tipos de dados que possuiam características próprias e importantes quando manipulados. As pesquisas envolvendo estes novos dados mostravam que as estruturas até então desenvolvidas não eram adequadas para a manutenção eficiente destes novos tipos de dados, como: séries temporais, sequências de proteínas, imagens, áudio, vídeo e textos longos, entre outros.

A estes novos dados deu-se o nome de dados complexos que tinham como característica marcante a ausência da relação de ordem total entre seus elementos. Por isso, estes dados não podiam ser indexados adequadamente usando as estruturas de dados existentes, obrigando assim que novos algoritmos e novas formas de indexação fossem pesquisadas.

Alguns conjuntos de dados complexos possuem uma característica importante que é o quanto um elemento se assemelha a outro. A partir dessa percepção viu-se que era possível a indexação de dados complexos baseado em um valor que media o quanto um elemento do conjunto era diferente de outro, valor este que foi denominado dissimilaridade. A dissimilaridade é um valor que mede o quanto dois elementos de um conjunto são (dis)similares, dessa forma, esses conjuntos de dados passaram a ser conhecidos como conjuntos de dados métricos [6]. A ideia desta nomenclatura está no fato de que dois elementos em um conjunto de dados métricos, por exemplo duas imagens, dificilmente serão iguais, portanto – como os elementos serão na sua grande maioria diferentes – é mais natural desvincular a noção de similaridade e usar a noção de dissimilaridade.

1.1 Motivação

Para manipular um conjunto de dados métricos é necessário uma forma de medir a dissimilaridade entre dois elementos o que pode ser feito usando uma função de distância métrica ou somente função métrica. Dado um conjunto de dados métricos *D*, uma função de distância é uma função capaz de computar a distância entre dois elementos deste conjunto. Uma função de distância *d()* retorna um valor $0 \leq d() \leq \infty$ onde o valor 0 (zero) equivale a dizer que os elementos são exatamente iguais e quanto maior o valor retornado por *d()* mais diferentes serão os elementos. A função de distância *d()* deve satisfazer as três regras do espaço métrico que são: *Não negatividade*: $d(x, y) \geq 0, \forall x, y \in D$, *Simetria*: $d(x, y) = d(y, x), \forall x, y \in D$, *Identidade*: $x = y \Leftrightarrow d(x, y) = 0, \forall x, y \in D$ e *Desigualdade triângular*: $d(x, z) \leq d(x, y) + d(y, z), \forall x, y, z \in D$ [3]. Sendo assim, a dissimilaridade ocorrerá na medida em que a distância entre os elementos cresce ou $d(a, b) \longrightarrow \infty$.

As funções de distância podem ser classificadas como discretas ou contínuas. Uma função de distância discreta retorna somente um pequeno conjunto de valores predefinidos, por exemplo, função de distância de *Levenshtein* [7]. A função de distância de *Levenshtein* retorna a quantidade de substituições, inserções ou remoções que devem ser realizadas em uma palavra para que ela se transforme

em outra. Por exemplo, *d(ana, asa) = 1*, pois, alterando a letra "n" por "s", a palavra "ana" vira "asa". Uma função de distância contínua retorna um conjunto de valores com cardinalidade muito grande tendendo ao infinito, por exemplo, função de distância entre vetores no espaço euclidiano [8].

Estas características das funções de distância em espaços métricos possibilitam a manipulação de dados métricos em estruturas de dados que permitam a sua indexação de uma forma hierárquica sem a necessidade da relação de ordem total. Os primeiros algoritmos permitiam operações unárias como a seleção, presente em qualquer sistema gerenciador de banco de dados (SGBD) moderno. Consultas como: "*Dado uma imagem, buscar as imagens mais semelhantes*", tornaram-se comuns e objeto de pesquisa.

Outro tipo de operação comum e alvo de muitas pesquisas em SGBD é a junção. A junção é uma operação binária capaz de combinar vários conjuntos a fim de obter um outro conjunto de resposta, isto porque a junção é uma seleção baseada em um predicado relacionado a um ou mais atributos aplicada ao produto cartesiano entre os conjuntos envolvidos [9]. Por manipularem, muitas vezes, conjuntos grandes de dados, é muito importante que se tenha cuidado na construção de operadores de junção, pois normalmente não se tem uma grande quantidade de memória principal disponível o tempo todo para as aplicações manterem todos os dados envolvidos na junção. Também não se pode contar com operações de leitura no disco constantes, porque este tipo de acesso é muito custoso e pode degradar rapidamente o desempenho da operação de junção, tornando-a muito lenta e inviável na maioria dos casos. Sendo assim, a construção de operadores de junção requer uma minimização no que se refere à quantidade de acessos em disco, bem como na quantidade de cálculos que devem ser executados a fim de decidir se um determinado elemento fara parte do conjunto de reposta.

Alguns exemplos de junção em espaço métrico são: *Junção por Abrangência* (*Range Join*), *Junção por Vizinhos mais Próximos* (*Nearest Neighbor Join*), *Junção por Proximidade* (*Closest Nearest Join*) e *Junção ao Redor* (*Join Around*) [4]. A junção por abrangência pode ser usada para responder problemas do tipo "dado dois conjuntos de pontos pertencentes a um mesmo domínio e uma distância máxima, buscar todos os elementos que estejam distantes uns dos outros no limite da distância máxima estabelecida". A junção por vizinhos mais próximos responde problemas como "dado dois conjuntos de pontos pertencentes a um mesmo domínio e uma quantidade arbitrária qualquer, buscar a quantidade arbitrária de pares de pontos mais próximos de um conjunto para cada objeto do outro conjunto". A junção por proximidade responde à consultas semelhantes à junção por vizinhos mais próximos, porém no exemplo anterior, considerando a quantidade arbitrária igual a 2 e considerando que um dos conjuntos possui 10 pontos, pode-se encontrar até 20 pontos como resposta; já na junção por proximidade, tem-se os 2 pontos com a menor distância entre eles. A junção ao redor responde consultas combinando propriedades de junção por abrangência e junção por vizinhos mais próximos,

assim, responde a consultas do tipo “dado dois conjuntos de pontos pertencentes a um mesmo domínio, uma distância máxima e uma quantidade arbitrária qualquer, buscar a quantidade arbitrária de pontos que estejam à distância máxima uns dos outros”.

1.2 Rota

Diversas áreas do conhecimento utilizam a ideia de rota de várias maneiras. Nos dicionários a palavra rota é definida como diretriz, direção, rumo, curso, roteiro, itinerário, caminho, via, senda, estrada, artéria, traçado, entre outras. O conceito de rota é, certamente, bem intuitivo de um modo geral, mas pode causar muita confusão se usado sem o devido cuidado.

Neste trabalho uma rota é um grafo conexo acíclico de grau máximo 2 com pesos em suas arestas que representa a distância entre um par de vértices. Toda rota possui um ponto inicial e um ponto final distintos. No caso onde o ponto inicial e final sejam o mesmo ponto, a rota é definida por um ponto somente. Dado dois pontos distintos dentro da rota, um caminho pode ser obtido através da distância calculada entre estes elementos pondendo assumir qualquer forma dependendo do espaço em que se trabalha, por exemplo, um segmento de reta em um plano, um arco em uma esfera etc.

Como exemplo tem-se rotas geradas por dispositivos que utilizam o *Sistema de Posicionamento Global* (*Global Positioning System - GPS*) que originam rotas baseadas em informações do posicionamento geográfico de cada ponto no globo terrestre.

É importante lembrar que, apesar de usar rotas formadas por pontos que possuem latitude e longitude, outros domínios também podem conter a noção de rotas com suas características e significados próprios, por exemplo, uma rota em um conjunto de perfis de usuário podendo indicar uma tendência de uso ou consumo; uma rota em um conjunto de proteínas podendo indicar a evolução de uma patologia ou de um disturbio; bem como a evolução de uma vacina ou de um biocombustível menos poluente.

1.3 Objetivo

O objetivo deste trabalho é a definição de um novo operador de junção que responderá as consultas do tipo: “Quais são as cidades que estão até 1 km da rota do circuito das malhas (Jacutinga – Ouro Fino – Monte Sião)”. Esta operação é realizada entre dois conjuntos: o conjunto de pontos do domínio e o conjunto de pontos da rota que representa um grafo conexo e acíclico de grau máximo 2.

Este trabalho explora a utilização de funções de distância sujeitas à geometria euclidiana e à geometria esférica.

1.4 Considerações Finais

Este capítulo apresentou alguns conceitos preliminares sobre este trabalho, tais como junção, espaço métrico, estruturas de indexação, tipos de dados em espaço métrico, medidas de desempenho de operadores de junção e, por fim, algumas considerações a respeito do que é rota.

No próximo capítulo será apresentado o estado da arte em que se encontram as pesquisas relacionadas a este trabalho. Inicialmente, serão abordados conceitos de métodos de acesso espacial e métrico. Em seguida, serão formalizados alguns conceitos em relação à junção tradicional, junção métrica e espacial. Por fim, serão apresentados trabalhos sobre a junção por similaridade, por abrangência, vizinhos mais próximos e uma abordagem mais atual sobre junção ao redor e espaços de utilização.

O restante desta dissertação está organizada da seguinte maneira: capítulo 3 apresenta um novo operador de junção chamado de junção canalizada, o capítulo 4 apresenta os experimentos usando a estrutura de dados métrica *Slim-Tree*. Finalmente, no último capítulo são apresentadas as conclusões e as sugestões de trabalhos futuros.

CAPÍTULO
2
Revisão Bibliográfica

2.1 Considerações Iniciais

Neste capítulo serão apresentados os conceitos envolvendo métodos de acesso espacial, métrico e junção. Um método de acesso é um mecanismo usado para prover acesso a um conjunto de dados. Entende-se por acesso uma forma de possibilitar a inserção, remoção e busca de um determinado elemento armazenado dentro de uma estrutura de dados [3].

Os métodos de acesso espacial são usados em muitos trabalhos relacionados à junção no domínio de dados espaciais. A *R-Tree* [2] é um método de acesso espacial muito usado em junções, inclusive em trabalhos no domínio de dados métricos. No entanto, o uso deste tipo de método de acesso para o operador de junção proposto por este trabalho fica comprometido, uma vez que estrutura de dados espaciais como a *R-Tree* estão limitadas à dimensionalidade que envolve o conjuntos de dados indexados, ou seja, quanto maior a dimensionalidade do conjunto, pior é o desempenho do método espacial, e também porque métodos de acesso espacial não podem indexar um domínio de dados adimensional. Para os conjuntos que não possuem uma dimensão definida (adimensionais), por exemplo, cadeias de proteínas e conjunto de palavras, são usados métodos de acesso métrico como a *Slim-Tree* [10]. O restante do capítulo trata dos operadores de junção e seus algoritmos.

A operação de junção é uma operação binária importantíssima em banco de dados [9]. É através dela, por exemplo, que é possível representar dinamicamente os relacionamentos existentes entre as diversas relações. A junção é o produto cartesiano entre duas relações aliado a uma consulta que envolve a comparação entre uma tupla de cada relação, que é denominado predicado de junção ou condição de junção. Pela natureza das operações que envolvem o operador de junção, pode-se dizer que a junção é uma das operações mais caras computacionalmente pois requer muitas vezes acessos constantes e intensivos às relações envolvidas. Sendo assim, a operação de junção é usada para combinar tuplas relacionadas de duas relações em uma única relação resposta. A relação desejada

entre as tuplas envolvidas é dada pelo predicado de junção. A relação resultante de uma junção é basicamente uma seleção sobre um subconjunto do produto cartesiano entre as outras duas relações.

Por sua importância e complexidade, a junção é alvo de muitos estudos e discussões, visto que ela consome muito tempo de processamento e demanda uma quantidade maior de acessos a disco do que outras operações em banco de dados. Por isso, propor implementações de junção significa preocupar-se com I/O (*input/output*), em referência aos acessos em disco e CPU (*Central Processing Unit*), em referência à quantidade de processamento (cálculos) gasto nas comparações entre as tuplas envolvidas.

2.2 Método de Acesso de Ordem Total

A *B-Tree* é um método de acesso de ordem total (*MAOT*) muito usado nos atuais sistemas de banco de dados e sistemas de arquivos. Desenvolvida por *Rudolf Bayer* e *Edward M. McCreight* em 1971 [5], faz uso da relação de ordem total entre os elementos para manipular de forma eficiente dados que possuam esta relação.

A ideia básica de *Bayer* era aumentar a quantidade de chaves dentro do nó [11] em relação às árvores binárias de busca. As árvores binárias de busca possuem uma chave por nó e um ponteiro apontanto a subárvore esquerda, que armazena valores menores do que a chave, e um ponteiro apontando a subárvore direita, que armazena valores maiores do que a chave. Dessa forma, com mais chaves por nó, a *B-Tree* é menor em altura do que árvores de busca binária e maior em largura, permitindo menos acessos em disco cada vez que um nível é visitado.

Sendo assim, a informação mais importante da *B-Tree* é a definição da sua ordem que indica o número máximo de chaves que um nó pode armazenar. Considere *n* a ordem de uma árvore *B-Tree*, então, para se ter um balanceamento ideal, seus nós internos devem conter no mínimo $n/2$ chaves. Dessa forma, ao inserir ou remover chaves, o valor mínimo deverá ser considerado e divisões ou fusões de nós devem ser realizadas para garantir este balanceamento.

2.2.1 Junção em Dados de Ordem Total

Formalmente, considere dois esquemas de relações *R* e *S*. A junção *theta* $R \bowtie_{r(a)\theta s(b)} S$ pode ser definida como sendo:

$$R \bowtie_{r(a)\theta s(b)} S \equiv \pi_{R \cup S}(R\sigma_{r(a)\theta s(b)}[R \times S])$$

Onde a junção $R \bowtie_{r(a)\theta s(b)} S$ é a projeção em $R \cup S$ de uma seleção sobre $r \times s$, o predicado requer que $r(a)\theta s(b)$ seja verdadeiro para cada atributo $R \cap S$ [12]. Nos domínios onde os conjuntos

possuem relação de ordem total, o operador θ pode ser um dos operadores relacionais da matemática ($=, \neq, <, >, \leq, \geq$). O resultado da relação Q para a operação de junção representada anteriormente pode ser então escrita como:

$$Q = \{t \mid t = rs \wedge r \in R \wedge s \in S \wedge t(a)\theta t(b)\}$$

Existem várias implementações que podem ser usadas para construir diversos tipos de junções. A seguir serão apresentados alguns exemplos de implementações para a operação de junção.

2.2.2 Algoritmo de Junção de Laços Aninhados (*Nested-Loops Join* - NLJ)

Esta é a implementação mais simples para a operação de junção [9, 13]. Cada relação envolvida é percorrida por um laço de repetição. O laço mais externo percorre cada tupla da primeira relação e para cada tupla lida, um laço interno percorre todas as tuplas da segunda relação envolvida na junção. O laço mais interno compara as tuplas recuperadas e verifica se elas satisfazem o predicado de junção; no caso de ser verdade, as tuplas são concatenadas e colocadas no *buffer* de saída para serem armazenadas na relação resposta, no caso da comparação ser falsa, a tupla lida pelo laço interno é descartada e a próxima tupla é lida e feita a comparação para todas as tuplas desta relação. O Algoritmo 2.1 mostra a implementação para $R \bowtie_{r(a)\theta s(b)} S$.

Algoritmo 2.1 Implementação do NLJ. [9]

```
1: para cada tupla s faça
2:    para cada tupla r faça
3:       se r(a)θs(b) então
4:          concatena r e s
5:          armazena na relação Q
```

O algoritmo de junção *NLJ* é implementado como uma junção *nested-block*, isto é, as tuplas são recuperadas em blocos da seguinte forma: a relação mais interna é lida toda em um único bloco de uma só vez. O número de blocos restantes são usados para ler as tuplas da relação mais externa. Todas as tuplas do bloco da relação mais interna são unidas às tuplas do bloco lido da relação mais externa de acordo com o resultado da avaliação do predicado de junção. Depois um novo bloco da relação mais externa é lido e o processo é realizado novamente. Note que, para melhorar o desempenho do Algoritmo 2.1, a relação com a maior cardinalidade deve ser escolhida para ser a relação mais interna, pois esta será lida de uma só vez.

2.2.3 Algoritmo de Junção por Fusão (*Sort-Merge Join* - SMJ)

O *SMJ* é executado em duas etapas [9, 13]. Na primeira delas, as relações envolvidas na junção são ordenadas em relação aos atributos do predicado de junção. Na segunda, cada relação é percorrida e as tuplas que satisfazem o predicado de junção são concatenadas e armazenadas na relação resposta. O Algoritmo 2.2 mostra como realizar *equijoins* usando *SMJ*. O uso mais comum de junção envolve as condições de junção apenas em comparações de igualdade, assim, quando a junção possui apenas o operador de igualdade = como operador de comparação, ela é chamada *equijoin* [12].

Algoritmo 2.2 Implementação da *equijoin* usando *SMJ*.[9]

```
 1: Passo1: Ordenar as relações
 2: ordenar R em r(a);
 3: ordenar S em s(b);
 4: Passo 2: Unir as relações
 5: ler a primeira tupla de R;
 6: ler a primeira tupla de S;
 7: para (cada tupla r) faça
 8:    enquanto (s(b) < r(a)) faça
 9:       ler a próxima tupla de S;
10:       se (r(a) = s(b)) então
11:          junção de r e s
12:          armazena na relação Q;
```

Vale ressaltar que o algoritmo *SMJ* depende do operador θ envolvido e também depende se o atributo envolvido no predicado de junção é um atributo chave ou não, pois no caso de ser um atributo simples podem ocorrer repetições de valores nas tuplas e obrigar o algoritmo a verificar várias vezes o mesmo conjunto de tuplas da relação mais interna. Por isso, o Algoritmo 2.2 deve ter implementado em seu laço interno uma forma de retornar ao início da iteração atual quando encontrar uma tupla com valor duplicado na próxima leitura da tupla do laço externo.

O algoritmo *SMJ* também pode ser usado para implementar a *outerjoin* [9]. Quando realizamos uma junção, normalmente, se uma tupla em uma relação não encontra uma tupla correspondente na outra relação, ela é ignorada e não é colocada no conjunto resposta para esta junção. Quando desejamos forçar que todas as tuplas de uma relação ou de ambas sejam colocadas no conjunto resposta da junção, usamos junções externas ou *outerjoin* [12]. O grande sucesso do *SMJ* está no fato dele reduzir o número de comparações entre tuplas, exceto nos casos citados anteriormente, ou seja, uma tupla da relação do laço externo não será comparada com as tuplas da relação do laço mais interno, caso não seja possível efetuar a junção entre ambas.

2.2.4 Algoritmo de Junção Hash (*Hash Join* - HJ)

Os algoritmos de junção *HJ* conseguem os mesmos resultados do *SMJ* processando as relações de outra forma [9, 13]. Todos os algoritmos desta família iniciam particionando as relações em conjuntos

disjuntos denominados *buckets*. O algoritmo *HJ* é executado em duas fases distintas. Na primeira, ele particiona as relações interna e externa envolvidas colocando as tuplas nos seus respectivos *buckets*, usando uma função de partição ou função *hash*. Nesta etapa, o algoritmo tenta isolar as tuplas da primeira relação em que pode ser feita a junção com uma dada tupla da segunda relação. Depois as tuplas da segunda relação são comparadas com um conjunto menor de tuplas da primeira relação, geralmente usando a função *hash* e buscando a partição onde a junção poderá ser realizada. Na segunda fase da junção, os *buckets* correspondentes das relações interna e externa são examinados aplicando-se o predicado a cada um dos objetos, criando o conjunto resposta. Os algoritmos utilizados possuem duas restrições: na primeira fase, cada elemento da entrada é colocado em apenas um *bucket* e, na segunda fase, cada *bucket* do conjunto interno é comparado com apenas um *bucket* do conjunto externo. É importante notar que se um determinado valor em uma das relações estiver fora dos limites dos valores dos subconjuntos gerados da outra relação, esta tupla poderá ser ignorada. Dentre os métodos de junção usando *hashing* [9] alguns utilizam a junção *hash* simples, outros a junção *hash* particionada, *loops* de junção *hash* etc.

O algoritmo junção *hash* simples (*Simple Hash Join – SHJ*) é o algoritmo mais simples da família de *hash-join*. O *SHJ* processa um *bucket* por vez enquanto tenta fazer o mínimo de partições necessárias e as fases de particionamento e junção são executadas simultaneamente, auxiliadas por arquivos de entrada, saída e tabelas *hash* que são geradas em cada fase. O Algoritmo 2.3 implementa a $R \bowtie_{r(a)\theta s(b)} S$.

Algoritmo 2.3 Junção *Hash* Simples [9].

```
1: para cada tupla s em S faça
2:    hash junção dos atributos s(b)
3:    coloca as tuplas na tabela hash baseadas nos valores hash;
4: para cada tupla r faça
5:    se r é uma entrada não vazia da tabela hash para S então
6:       hash junção dos atributos r(a)
7:       se rθ - combina qualquer s no bucket então
8:          concatena r e s
9:          coloca na relação Q;
```

No Algoritmo 2.3, a mesma função *hash* deve ser usada para ambas as relações. O método de junção *hash* pode ser usado para implementar também as junções *right/left outerjoins* [9] como mostrado no Algoritmo 2.4.

No algoritmo de junção *hash* particionada (*Hash-Partitioned Join – HPJ*), as fases de particionamento e junção são executadas separadamente. Na fase de particionamento, ambas as relações são particionadas usando a mesma função *hash*. Usando uma abordagem "*dividir para conquistar*" o algoritmo *HPJ* divide o problema em subproblemas que podem ser processados de forma independente e, por este motivo, podem também ser paralelizados, uma vez que as tuplas de uma determinada par-

Algoritmo 2.4 Junção *Right Outerjoin* usando *SHJ* [9].

```
1:  para (cada tupla s em S) faça
2:     hash junção dos atributos s(b)
3:     coloca tuplas na tabela hash baseada nos valores hash;
4:  para (cada tupla r) faça
5:     se r é uma entrada não vazia da tabela hash para S então
6:        hash junção dos atributos r(a)
7:        se (ajusta s encontrado) então
8:           concatena r e s
9:           coloca na relação Q
10:       senão
11:          pad atributos de S com valores nulos;
12:          escrever r em Q
```

tição R_i ($i \in R$) na relação R, só podem ser comparadas às tuplas correspondentes à mesma partição S_i ($i \in S$) na relação S.

O algoritmo de junção de *hash* particionada simples (*Simple Hash-Partitioned Join – SHPJ*) gera uma partição e a usa antes de gerar a próxima. Nesta abordagem, duas tabelas temporárias vão sendo criadas, na medida em que as tuplas que não se casam com a partição que está sendo analisada são encontradas. Depois disso, as relações temporárias são usadas como as relações a serem analisadas para a próxima partição que será criada. Isto impede que as tuplas que já foram colocadas no arquivo de saída sejam analisadas novamente. O Algoritmo 2.5 mostra como realizar $R \bowtie_{r(a)\theta s(b)} S$.

Algoritmo 2.5 Junção de *Hash* Particionada Simples [9].

```
1:  repita
2:     escolher uma área hash para partição atual
3:     para (todo s) faça
4:        valores dos atributos hash;
5:        se (valor cai na faixa corrente) então
6:           assuma a entrada na tabela de hash particionada
7:        senão
8:           escrever a tupla em temp-S;
9:     para todo r faça
10:       hash nos atributos da juncao;
11:       se valor cai na faixa corrente então
12:          se correspondência foi encontrada na tabela hash então
13:             concatena as tuplas r e s
14:             coloca na relacao saida Q
15:          senão
16:             descarte a tupla
17:       senão
18:          escrever a tupla em temp-R;
19:    R := temp-S
20:    S := temp-R
21: até que (| S |= null);
```

O *loops* de junção *hash* (*Hash Loops Join – HLJ*) é uma variação do Algoritmo 2.1 apresentado. Nesta abordagem, durante a primeira fase, a relação externa é dividida em um número fixo de partições. No passo seguinte, cada partição é lida na memória. Uma tabela *hash* é gerada para cada partição quando ela é lida da memória principal. Depois disso, as tuplas na relação interna são lidas e os atributos de junção da relação interna são mapeados. Os valores mapeados são usados para buscar

na tabela *hash* as partições que possam combinar as tuplas. A relação interna será lida uma vez para cada partição gerada e a tabela *hash* é usada para acelerar o processo de encontrar uma combinação. O Algoritmo 2.6 mostra como realizar $R \bowtie_{r(a)\theta s(b)} S$.

Algoritmo 2.6 *Loops* de Junção *Hash* [9].

```
1: particiona R usando a função de divisão hash;
2: para cada partição Ri faça
3:    ler a partição em memória;
4:    criar a tabela hash para a partição;
5: para cada tupla s faça
6:    hash valor do atributo;
7:    se s hashes para uma entrada não vazia da tabela hash para Ri então
8:       iniciar a junção de s e r;
9:    coloca o resultado em Q;
```

2.3 Métodos de Acesso Espacial

Durante os últimos anos, a quantidade de informações coletadas por satélites, sondas e outros dispositivos fizeram com que muita informação fosse gerada com relação à superfície da Terra, posicionamento geográfico etc. Com essa grande quantidade de informação geográfica, surgiram os Sistemas Gerenciadores de Banco de Dados Geográficos (*SGBDG*) que utilizam métodos de acesso espacial para manipular estas informações.

Um *Método de Acesso Espacial* ou *MAE* é um mecanismo envolvendo algoritmos e estrutura de dados capaz de indexar e acessar dados que existam no domínio de dados espaciais. O *MAE* mais conhecido e referenciado na literatura é a *R-Tree* [5, 13, 1] e suas variantes, R^**-Tree* [14] e R^+*-Tree* [15].

A *R-Tree* é uma estrutura altamente balanceada e dinâmica que possui uma organização estrutural interna semelhante à *B+-Tree* [5, 13]. Por ser balanceada, todo nó folha possui a mesma altura em relação à raiz com crescimento de baixo para cima (*bottom up*). Por ser dinâmica, pode-se inserir e remover objetos da estrutura sem que haja a necessidade de reconstruir toda a árvore novamente [1].

O nó da *R-Tree* corresponde a uma página no disco que armazena um conjunto de objetos geométricos e contém entradas na forma (ref, ret) onde *ref* é o endereço do nó filho e *ret* é o menor retângulo possível que consegue englobar o objeto espacial indexado (*Minimum Bounding Rectangle - MBR*). Um *MBR* é limitado pelo intervalo fechado [*min*,*max*] que representa a extensão do objeto sobre uma dimensão. Alternativamente, o intervalo pode ter um ou ambos os pontos igual a ∞, indicando que o objeto se estende indefinidamente, por exemplo, no início da construção o *MBR* possui dimensão ∞ até que ocorra a primeira divisão do nó. Os nós folhas da *R-Tree* contêm ponteiros para os objetos armazenados e o *MBR* do objeto espacial.

A grande vantagem na utilização de *MBR* na *R-Tree* é que a complexidade na representação geométrica dos objetos indexados é reduzida a dois pontos, sua posição e suas dimensões que são as características mais importantes dos objetos indexados. Considere *M* sendo o número máximo de entradas que o nó suporta e o número mínimo de entradas no nó é de $m = M/2$. A *R-Tree* deve satisfazer as seguintes propriedades:

1. Todo nó folha contém entre *m* e *M* registros, exceto se for raiz;

2. Para cada registro índice (*ref, ret*) em um nó folha, *ref* é o menor retângulo que contém espacialmente o objeto de dado representado pela tupla mencionada;

3. Todo nó não folha possui entre *m* e *M* filhos, exceto o nó raiz;

4. Para cada entrada (*ref, ret*) em um nó folha, *ref* é o menor retângulo que contém espacialmente os retângulos no nó filho;

5. O nó raiz tem no mínimo dois filhos exceto se for um nó folha;

6. Todos os nós folha aparecem no mesmo nível.

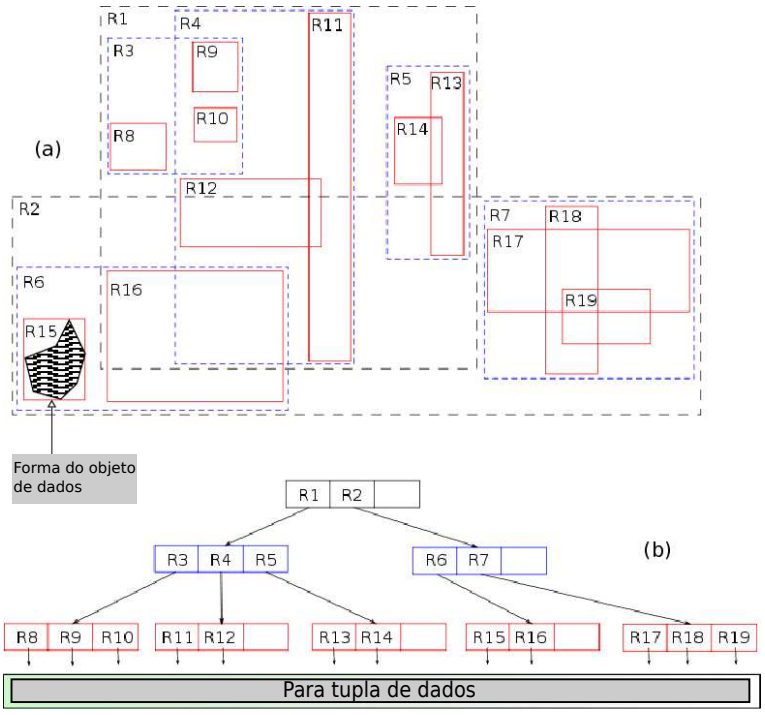

Figura 2.1: Representação espacial (a) e gráfica (b) de uma R-tree [1].

A Figura 2.1 (a) ilustra as relações de contenção e sobreposição que podem existir em uma *R-Tree* e a Figura 2.1 (b) mostra a estrutura da *R-Tree* formada após a inserção de vários objetos espaciais.

As inserções de novos objetos na *R-Tree* ocorrem sempre em nós folha. Quando um nó possui a sua capacidade excedida, o nó em questão deve ser dividido e os objetos devem ser redistribuídos entre os novos nós, isso é conhecido como *split* do nó. Os *splits* subsequentes, se houverem, são propagados para cima em direção à raiz e, por fim, se houver um *split* na raiz a árvore ganhará um nível.

Para inserir um novo objeto na *R-Tree*, o nó folha onde este novo objeto será inserido é localizado. Caso mais de um nó se qualifique para receber o novo objeto, o nó que possui o menor *MBR* é escolhido e o objeto é inserido. Caso não seja encontrado um nó que possa receber o objeto, o nó que possui um *MBR* que possa englobar a maior área do objeto é escolhido e este *MBR* é então ajustado. Todas as alterações necessárias são propagadas para os nós acima além dos *splits*, caso existam.

2.3.1 Junção em Dados Espaciais

Quando se trabalha o conceito de junção espacial chega-se à constatação que, de um modo geral, as abordagens existentes consistem basicamente de algoritmos que executam o particionamento dos dados e a junção espacial no subconjunto de dados verificando se existem interseções. Baseado nestes princípios, muitos Sistemas de Informações Geográficas (*SIG*) têm sido desenvolvidos, principalmente, para o estudo geográfico e estatístico, cartografia, ciências naturais, ambientais, engenharias, biologia etc. Os objetos manipulados em cada uma destas áreas são objetos multidimensionais compostos por pontos, linhas, superfícies e atributos como área, volume, e em alguns casos, um atributo temporal [16].

Uma junção espacial busca todos os pares de objetos que satisfazem uma relação espacial entre os objetos envolvendo valores de seus atributos espaciais, como uma interseção [17] ou uma sobreposição. Por exemplo, uma consulta complexa seria “procurar as cidades que possuam áreas de vegetação nativas com taxa de desmatamento acima de 30%, nos últimos 5 anos, no norte do Brasil”. De forma simplificada, contando com um mapa de cidades no Brasil, um mapa de cobertura do solo, dados históricos de cobertura para calcular a taxa de desmatamento, localiza-se as áreas de cidades que se sobrepuserem às áreas de vegetação com a taxa de desmatamento desejada, estas áreas que se intersectam conterão as cidades que solucionam o problema. Pode-se estender a ideia de sobreposição como relação entre os objetos de várias outras formas como proximidade, fechamento, no sentido de envolver, relações de direção etc. Além disso, existe a junção espacial de um conjunto, um *self-spatial join*, ou uma junção com mais de dois conjuntos envolvidos, uma *multiway spatial join*.

Tomando como base que a definição de junção espacial pode ser considerada uma extensão da definição de junção formal dada na subseção 2.2, sugere-se a utilização dos algoritmos básicos de junção para a execução da junção espacial. Isto pode ser verdade até certo ponto. Considere usar

a abordagem de *loops* aninhados varrendo as relações envolvidas comparando cada elemento das relações para testar se o predicado de junção é satisfeito, então esta abordagem está correta. Como a quantidade de objetos destes conjuntos pode se tornar muito grande. Isto pode afetar o sistema por completo. Na verdade, operações de junção em banco de dados geográficos são mais comuns do que operações de seleção em *SIG* e, como geralmente o tempo de execução de uma junção espacial é superlinear no número de objetos dos conjuntos envolvidos[2], esta operação pode tornar-se um ponto de desempenho crítico.

Na subseção 2.2.1 foram apresentados alguns exemplos de implementações de algoritmos para junção. Na verdade, somente o algoritmo *nested-loop* 2.2.2 pode ser usado em junções espaciais sem nenhuma alteração. A junção *sort-merge* mostrada na subseção 2.2.3 precisa de uma ordenação unidimensional dos objetos para ser usada, logo, existe a necessidade de um ajuste na representação dos dados para poder usar esta abordagem. A Figura 2.2 (*b*) mostra um exemplo de ordenação usando a projeção dos objetos no eixo *x*. Por fim, as dificuldades de usar os algoritmos baseados em *hash* apresentados na subseção 2.2.4 podem ser resumidas em dois pontos principais [18]. Em primeiro lugar, a natureza do predicado da junção espacial é diferente da natureza do predicado da junção relacional. Geralmente, na abordagem relacional, busca-se um valor igual para o atributo ou atributos envolvidos no predicado de junção. Logo, sempre que $x = y$, tem-se também, $f(x) = f(y)$. Isso garante que tuplas com mesmo valor para o atributo escolhido serão mapeadas para o mesmo *bucket*. Já na abordagem espacial, a busca por dois polígonos exatamente iguais é muito remota e muitas vezes sem utilidade prática. Na verdade buscam-se polígonos que possuam alguma interseção. Por isso, dados dois polígonos P_1 e P_2 com $P_1 \neq P_2$, que é o caso padrão em qualquer conjunto de dados espacial, não é possível garantir que $f(P_1) = f(P_2)$ por melhor que seja $f()$ já que $P_1 \neq P_2$. Em segundo lugar, devido ao fato de que os dados espaciais não possuem relação de ordem total, a maioria das funções de ordenação total para dados espaciais não preservam uniformemente a proximidade espacial [19]. Segundo *Gaede*, isso permite que dois objetos adjacentes no espaço sejam mapeados distantes um do outro durante a ordenação.

A técnica mais comum utilizada é a junção espacial usando a *R-Tree* ou *R-Tree Spatial Join* (RSJ) [2]. Nesta abordagem, cada conjunto de dados R e S é indexado por uma *R-Tree*. A *RSJ* se baseia na propriedade do limite mínimo, onde a distância entre dois pontos nunca é maior do que a distância entre as regiões onde os pontos estão inseridos. O algoritmo *RSJ* percorre os índices de R e S simultaneamente, quando um par de páginas está sendo analisado, o algoritmo forma todos os pares de páginas filhas das páginas em questão cuja distância não é maior do que um valor ε, estes pares de páginas filhas são então percorridas recursivamente procurando aquelas que fazem parte do conjunto resposta. Trabalhos sugeriram alterações na *R-Tree* a fim de melhorar o seu desempenho

podendo melhorar o desempenho da *RSJ*. Dentre estas melhorias, existe a R^**-Tree* [14] que trata da inserção de novos objetos de forma diferente realocando estes novos objetos antes de particionar um nó, e o algoritmo de particionamento visa minimizar o perímetro das regiões e maximizar a ocupação dos novos nós contribuindo para diminuição da quantidade de acessos a disco durante a junção. Na R^+*-Tree* [15], tenta-se minimizar a sobreposição dos nós quebrando os objetos e armazenando suas partes em nós diferentes. Isso faz com que divisões no nó tenham que ser propagadas para o nó pai e para sua subárvore e, no caso do particionamento da subárvore, pode reduzir a taxa de ocupação da árvore como um todo [20], piorando a junção.

2.3.2 Algoritmo de Junção Espacial para R-Tree

A primeira abordagem proposta por *Kriegel* [2] é bastante intuitiva. Usando a propriedade de que os retângulos *MBR* direcionam para as suas respectivas subárvores, caso os retângulos de duas entrada E_r e E_s não possuam nenhuma interseção, neste caso, não existirá nenhum par $(rect_R, rect_S)$ de retângulos de dados que se intersectam, onde $(rect_R$ está na subárvore E_r e $rect_S)$ está na subárvore E_s. Caso os retângulos de duas entradas E_r e E_s possuam interseção, então as subárvores deverão ser visitadas à procura de outras interseções, se existirem. A Figura 2.2 (a) mostra um exemplo de interseção entre regiões de duas *R-Tree*, *r* e *s*. O Algoritmo 2.7 mostra a implementação para a abordagem apresentada considerando a altura de *r* e *s* iguais.

Algoritmo 2.7 JunçãoEspacial_01(R, S: R_Node)

```
1: para cada E_S ∈ S faça
2:    para cada E_R ∈ R com E_R.rect ∩ E_S.rect ≠ ∅ faça
3:       se R é FOLHA então
4:          // S também é folha
5:          adiciona o par E_R e E_S na resposta
6:       senão
7:          R' ← bloco referenciado em E_R.ref
8:          S' ← bloco referenciado em E_S.ref
9:          JunçãoEspacial01(R',S')
```

Assuma que *R* e *S* são nós da *R*-Tree* e que cada nó possui entradas *E* onde cada entrada possui um ponteiro *ref* para a subárvore apontada por ele e um retângulo *rect* n-dimensional representando o *MBR*. O algoritmo percorre as árvores envolvidas de modo *top-down* verificando a condição de junção para cada entrada do nó de uma árvore com todas as entradas do nó da outra árvore.

O Algoritmo 2.8 é uma melhoria do Algoritmo 2.7 pois usa a técnica de restrição de espaço de busca que consiste em usar um retângulo de interseção entre os nós de um determinado nível para restringir os espaços de busca do nível imediatamente abaixo.

No Algoritmo 2.8, *rect* é o retângulo de interseção *MBR* dos nós *R* e *S* e é usado na iteração seguinte para restringir o espaço de busca para todas as entradas de *R* e *S*.

Algoritmo 2.8 JuncaoEspacial_02(R, S: R_Node; rect:Retangulo)

```
1:  para cada E_S ∈ S com E_S.rect ∩ rect ≠ ∅ faça
2:    para cada E_R ∈ R com (E_R.rect ∩ rect ≠ ∅) e (E_R.rect ∩ E_S.rect ≠ ∅) faça
3:      se R é FOLHA então
4:        // S também é folha
5:        adiciona o par E_R e E_S na resposta
6:      senão
7:        R' ← bloco referenciado em E_R.ref
8:        S' ← bloco referenciado em E_S.ref
9:        rect ← retangulo de interseção das MBRs do nós R' e S'
10:       JunçãoEspacial02(R', S', rect)
```

A segunda abordagem proposta por *Kriegel* para melhorar a junção espacial é ordenar as entradas do nó da *R*-Tree* de acordo com as localizações espaciais dos retângulos correspondentes. O problema que ocorre nesta técnica é que retângulos bi-dimensionais não podem ser ordenados sem que haja a perda de sua localização. Entende-se aqui que, para serem ordenados, retângulos bi-dimensionais devem ser mapeados para uma sequência unidimensional. A solução adotada, segundo *Brinkhoff*, foi usar a interseção da seguinte forma: considere uma sequência $R_{seq} = \langle r_1, ..., r_n \rangle$ de *n* retângulos. O retângulo r_i é determinado pelo seu canto inferior esquerdo $(r_i.x_1, r_i.y_1)$ e seu canto superior direito $(r_i.x_u, r_i.y_u)$. A notação $\pi_X(r_i)$ e $\pi_Y(r_i)$ é usada para se referir à projeção de r_i nos eixos *X* e *Y* respectivamente. A sequência $R_{seq} = \langle r_1, ..., r_n \rangle$ é ordenada em relação ao eixo *X* se $r_i.x_1 \leq r_{i+1}.x_1, 1 \leq i < n$. Por exemplo, uma sequência ordenada dos seis retângulos da Figura 2.2 (b) seria $\langle s_1, r_1, r_2, s_2, r_3, s_3 \rangle$.

Algoritmo 2.9 TesteIntersecaoOrd(Rseq,Sseq:SeqRetang; Saida:ParesSeqRetang) [2]

```
1:  // Rseq e Sseq estão ordenados; ‖ Rseq ‖ número de retângulos em Rseq
2:  Saida:=; i = 1; j = 1;
3:  enquanto (i ≤‖ Rseq ‖) ∧ (j ≤‖ Sseq ‖) faça
4:    se (r_i.x1 < s_j.x1) então
5:      // t = r_i
6:      LoopInterno(r_i, j, Sseq, Saida);
7:      i = i + 1;
8:    senão
9:      //t = s_j
10:     LoopInterno(s_j, i, Rseq, Saida);
11:     j = j + 1;
```

Algoritmo 2.10 LoopInterno (t:Retangulo; desmarc:int; Sseq:SeqRetang; Saida:ParesSeqRetang) [2]

```
1:  K = desmarc;
2:  enquanto (k ≤‖ Sseq ‖) ∧ (s_k.x1 ≤ t.xu) faça
3:    // interseção X
4:    se (t.y1 < s_k.yu) ∧ (t.yu > s_k.y1) então
5:      // intersecao Y
6:      Append(Saída(t, s_k))
7:    k = k+1;
```

Uma outra abordagem apresentada é chamada *plane sweep* ou varredura do plano. A ideia básica desta abordagem é mover uma linha chamada linha de varredura perpendicularmente a um dos eixos, tome por exemplo o eixo *X*, a linha será movida da esquerda para a direita. Suponha duas sequências

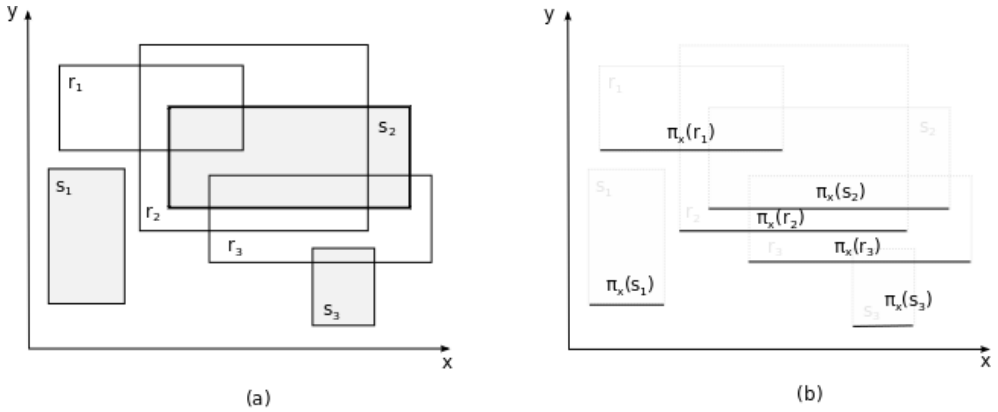

Figura 2.2: Interseção entre retângulos (a) na *R-Tree* e a projeção no eixo x (b) [2].

de retângulos R_{seq} e S_{seq}, a técnica de varredura do plano é usada sem o custo da criação de qualquer outra estrutura de dados dinâmica da seguinte maneira: sejam $R_{seq} = \langle r_1, ..., r_n \rangle$ e $S_{seq} = \langle s_1, ..., s_m \rangle$ sequências de retângulos. Primeiro, R_{seq} e S_{seq} são ordenados como descrito acima, assim tem-se, $r_i.x_1 \leq r_{i+1}.x_1, 1 \leq i < n$ e $s_j.x_1 \leq s_{j+1}.x_1, 1 \leq j < m$. Move-se, então, a linha de varredura para o retângulo *t* em $R_{seq} \cup S_{seq}$ com o menor valor de x_1. Se o retângulo é *t* em R_{seq}, percorre-se sequencialmente S_{seq}, iniciando a partir do seu primeiro retângulo, até um retângulo s_h em S_{seq} ser encontrado onde o valor de x_1 seja maior que $t.x_u$. Assim, obtém-se o intervalo $\pi_X(t)$ que intersecta o intervalo $\pi_x(s_j)$ para todo *j* com $1 \leq j < h$. Se além disso, o intervalo π_y também intersecta o intervalo $\pi_y(s_j)$, então o retângulo *t*também intersecta o retângulo s_j. Se o retângulo *t* está em R_{seq}, R_{seq} por analogia é atravessada. Após isso, o retângulo *t* é marcado para ser processado. Assim, a linha de varredura é movida para o próximo retângulo em $R_{seq} \cup S_{seq}$ desmarcado com o menor valor para x_1 e os passos descritos acima são repetidos para todos os retângulos desmarcados. Quando a última entrada para *R* ou *S* tiver sido processada, todas as interseções terão sido encontradas. O algoritmo 2.11 mostra um exemplo para esta abordagem.

Algoritmo 2.11 JuncaoEspacial03 (R,S:R_Node; rect:Retang) [2]

```
 1: R = {E1 | (E1 ∈ R) ∧ (E1.rect ∩ rect ≠ ∅)}
 2: S = {E1 | (E1 ∈ S) ∧ (E1.rect ∩ rect ≠ ∅)}
 3: OrdenaRetang (R);
 4: OrdenaRetang (S);
 5: TesteIntersecaoOrd(R,S,Seq);
 6: para i = 1 ate || Seq || faça
 7:    (E_R, E_S) = Seq[i]
 8:    se R e folha então
 9:       // S também e folha
10:       adiciona o par E_R e E_S na resposta
11:    senão
12:       R' ← bloco referenciado em E_R.ref
13:       S' ← bloco referenciado em E_S.ref
14:       rect ← retângulo de interseção das MBRs do nós R' e S'
15:       JunçãoEspacial03(R', S', rect)
```

O algoritmo de varredura do plano determina as interseções de dois conjuntos de retângulos. Ba-

seado na ordenação espacial, este algoritmo cria uma sequência de interseção de pares de retângulos. Esta sequência pode ser usada também para determinar o planejamento da leitura na junção espacial. Além disso, preserva a localidade espacial no *buffer* e pode ser usada sem qualquer custo extra. Esta abordagem é chamada ordenação por varredura de plano local, do inglês "*local plane-sweep order*" e corresponde ao Algoritmo 2.11.

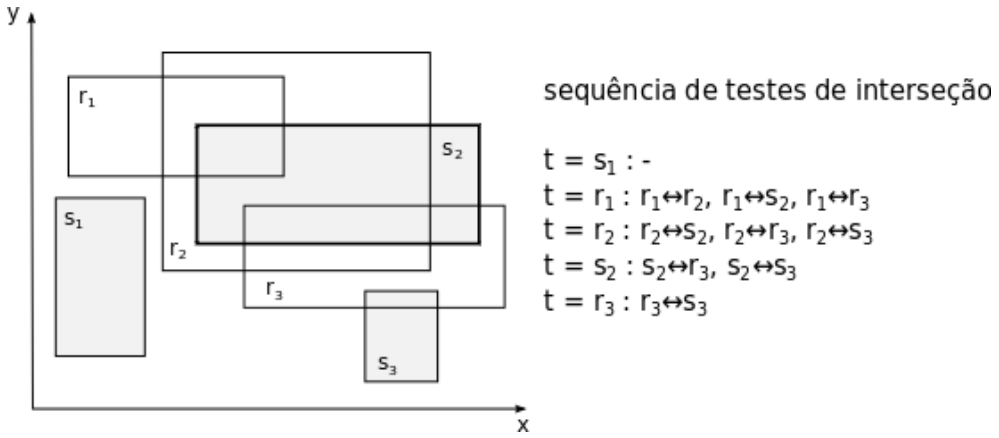

Figura 2.3: Teste de interseção ordenada [2].

A Figura 2.3 mostra um exemplo de como o algoritmo se comporta para os testes. O gráfico mostra o conjunto de retângulos, a linha de varredura irá parar nos retângulos s_1, r_1, r_2, s_2, r_3 respectivamente. Para cada parada, os pares de retângulos que serão testados para a interseção são mostrados à direita do gráfico.

Existem ainda duas outras variações de ordenação baseadas em varreduras de planos (*plane-sweep*) que são a ordenação por varredura de planos locais fixados: primeiramente, o algoritmo é igual ao de planos locais que determina um par de entradas (E_R, E_S). Após processar as sub-árvores E_R.ref e E_S.ref, é computado o grau do retângulo de ambas as entradas. O grau de um retângulo de uma entrada *E* é dado pelo número de interseções entre os retângulos *E.rect* e os retângulos que pertencem às entradas da outra árvore não processadas até o momento. Segundo, fixa-se a página no *buffer* com o retângulo que tem o grau máximo. Terceiro, a junção é realizada na página fixada com todas as outras páginas. Então, determina-se o próximo par de entradas usando novamente a ordenação por varredura de planos locais. Na outra variação de ordenação baseada em varreduras de planos, chamada ordenação em *Z* local, a ideia básica é realizar a ordenação através dos centros dos retângulos representados por pontos bidimensionais. Uma técnica para ordenar pontos bidimensionais é baseada em curvas de preenchimento de espaço, por exemplo curvas de *Peano*, também chamada de ordenação-Z. Essa técnica consiste em decompor o espaço em células de tamanho igual e realizar a ordenação nesse conjunto de células.

2.4 Métodos de Acesso Métrico

Com a grande popularização da internet no final da década de 80 e o acesso fácil à dispositivos modernos de captura e armazenamento, por exemplo, dispositivos móveis de comunicação, a quantidade de dados multimídia gerados vêm crescendo a cada dia. Junta-se a isso a utilização de imagens na área de saúde, séries temporais, cadeias de proteínas, documentos cada vez mais complexos e tem-se conjuntos de dados que não podem ser indexados de forma eficiente pelos SGBD relacionais atuais e, na maioria das vezes, também não podem ser indexadas por sistemas que usam estruturas espacias, como os casos de cadeias de proteínas e conjunto de palavras [13].

É importante perceber que no domínio de dados métricos não existe a relação de ordem total, por isso, operações que envolvam operadores relacionais ($=, \neq, <, >, \leq, \geq$) não fazem nenhum sentido prático neste domínio. O que existe então, é somente a relação de semelhança entre os objetos, ou seja, dados dois objetos, existe uma função de distância que retorna a dissimilaridade entre eles. Muitas estruturas foram propostas no sentido de indexar dados métricos. Estas estruturas se baseiam na dissimilaridade entre os objetos para responder a consultas por similaridade.

Um exemplo de *Método de Acesso Métrico MAM* é a *Slim-Tree* [3], que nasceu como uma extensão da *M-Tree*[21], com o objetivo principal de tentar reduzir as sobreposições entre as regiões de cobertura que prejudicava muito o desempenho destes tipos de estruturas, reduzindo também o seu tempo de construção.

A *Slim-Tree* é uma estrutura métrica dinâmica implementada em disco cujo crescimento ocorre de baixo para cima *bottom up*. Isso indica que as inserções são sempre feitas em nós folha e as divisões dos nós (*splits*) fazem com que a árvore seja atualizada em direção ao nó raiz, o que garante uma estrutura naturalmente balanceada.

Na *Slim-Tree* os objetos são agrupados como páginas de tamanho fixo em disco e cada página representa um nó na árvore fazendo com que cada nó comporte um número máximo de objetos predefinido. Este tipo de estrutura permite que os objetos estejam organizados de forma hierárquica que usa um objeto representativo como o centro de cada região mínima, limitada e definida pelo raio de cobertura deste objeto, e que "engloba"(ou cobre) os objetos na subárvore.

A *Slim-Tree* possui dois tipos de nós. Nós de dados chamado folhas e nós de índices chamados nós internos. A Figura 2.4 ilustra os dois tipos de nós da *Slim-Tree*. Nós folha contêm todos os objetos armazenados na *Slim-Tree* e sua estrutura é mostrada a seguir:

$$noFolha[arrayOf < Oid_i, d(S_i, S_{rep}), S_i >]$$

Onde: Oid_i é o identificador do objeto, S_i e $d(S_i, S_{rep})$ é a distância entre o objeto S_i e o objeto

representativo do nó folha S_{rep}. A estrutura do nó indice é mostrada a seguir:

$$noIndice[arrayOf < S_i, R_i, d(S_i, S_{rep}), Ptr(TS_i), NEntries(Ptr(TS_i)) >]$$

Onde: S_i mantém o objeto representativo da subárvore apontada pelo ponteiro $Ptr(TS_i)$ e R_i é o raio de cobertura da região. A distância entre S_i e o representativo do nó S_{rep} é guardada em $d(S_i, S_{rep})$. O ponteiro $Ptr(TS_i)$ aponta para o nó raiz da subárvore representada por S_i. O número de entradas no nó apontadas por $Ptr(TS_i)$ é armazenado por $NEntries(Ptr(TS_i))$ e é usado pelo algoritmo *Minoccup* para garantir uma taxa alta de ocupação dentro do nó, a fim de minimizar a quantidade de acessos ao disco e também auxilia o algoritmo *slim-down* no cálculo do *fat-factor*, que será mostrado mais adiante.

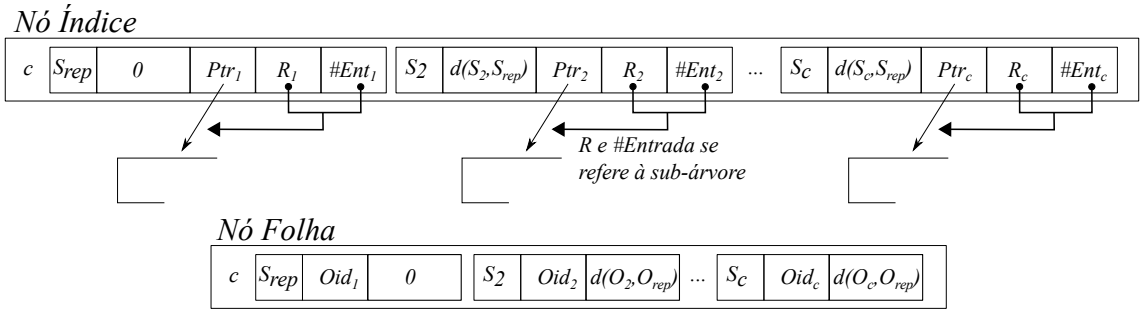

Figura 2.4: Representação gráfica dos nós da *Slim-Tree* [3].

A construção da *Slim-Tree* é auxiliada por diversos algoritmos. Incialmente, para inserir um objeto na *Slim-Tree*, o algoritmo procura a partir do nó raiz um nó cujo raio de cobertura englobe o objeto que está sendo inserido. Caso não exista um nó que possa fazê-lo, procura-se um nó cujo centro está mais próximo do objeto que está sendo inserido. No caso onde mais de um nó se qualifique, o algoritmo *choose subtree* é usado para decidir qual subárvore será navegada para que o objeto seja inserido. A *Slim-Tree* provê três opções para o algoritmo *choose subtree*: *random* escolhe aleatoriamente um dos nós que se qualificarem. É mais rápido, porém não garante a taxa de utilização dos nós. O *min dist* escolhe o nó cujo representativo possui a menor distância para o objeto que está sendo inserido. Por fim, o *min occup* escolhe o nó com a menor ocupação para inserir o objeto.

Na *Slim-Tree* quando um nó excede sua capacidade ele é dividido, o que é chamado de *split*, como na *R-Tree*. Quando isto ocorre, um novo nó é alocado e os objetos são redistribuídos e se elege representativos que serão promovidos. Caso o nó raiz sofra uma divisão, a árvore aumentará seu tamanho e ganhará um nível. A *Slim-Tree* conta com três algoritmos para auxiliar na execução da divisão de nós quando esta ocorre. Na primeira abordagem, o algoritmo *random* escolhe aleatoriamente dois objetos para serem os novos representativos. Apesar de ser um método rápido esta abordagem pode gerar nós com grandes raios de cobertura e fazer com que apareçam muitas regiões que se intersectam

na árvore prejudicando seu desempenho como um todo. O algoritmo *min max* testa todos os pares possíveis como potenciais representativos. O par que minimizar o raio de cobertura é escolhido. Esta abordagem é utilizada na *M-Tree* original [21] e possui complexidade $O(C^3)$, usando cálculos de distância, a complexidade passa a ser $O(C^2)$. Por último, o algoritmo *MST*, do inglês *Minimal Spanning Tree*. Nesta abordagem, um grafo com todos os objetos envolvidos é construído onde cada entrada dos nós são os vértices e a distância entre eles são as arestas do grafo. A construção da árvore geradora mínima garante a presença das menores arestas, consequentemente, a presença dos menores raios de cobertura possíveis. Depois, a aresta mais apropriada é removida da *MST* levando-se em conta a taxa de ocupação. As duas subárvores resultantes são assumidas como a nova configuração dos nós da *Slim-Tree*. Para finalizar, são escolhidos dois objetos representativos dentro de cada novo grupo que possua o menor raio de cobertura em relação a todos os outros objetos. As atualizações necessárias são então realizados nos novos nós e propagadas para os nós superiores em direção à raiz.

A *Figura* 2.5 (*a*) apresenta graficamente uma árvore *Slim-Tree* e (*b*) a visualização espacial desta estrutura. Nota-se que os objetos localizados nas regiões de interseção, como o caso do objeto *O*, forçam que uma consulta tenha que visitar os dois nós que se soprepõem para saber em qual deles está o objeto.

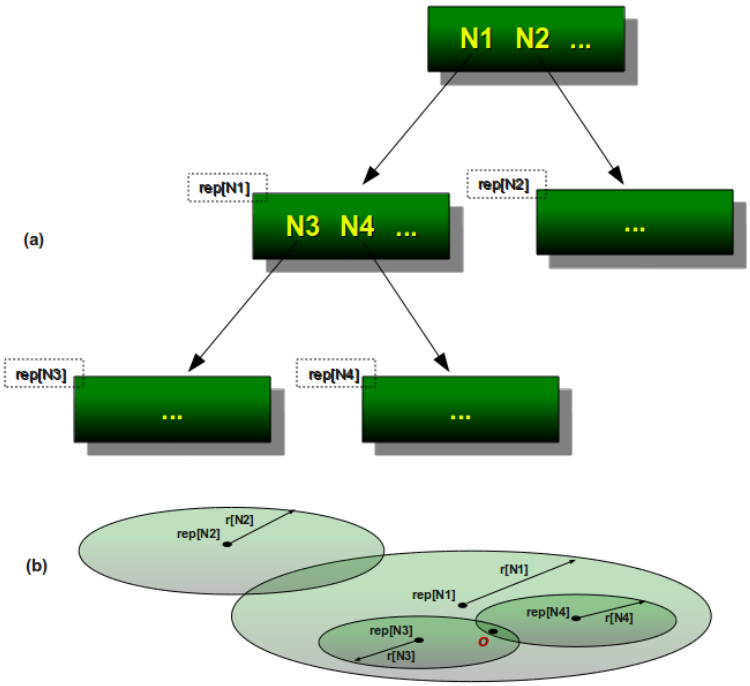

Figura 2.5: Representação gráfica (a) e espacial (b) da *Slim-Tree*.

Durante a fase de construção da *Slim-Tree* muitas regiões podem tornar-se muito grandes fazendo com que apareçam muitas áreas que se sobrepõem umas as outras. Estas áreas em comum são o principal motivo na degradação no desempenho da *Slim-Tree*. Para melhorar o seu desempenho deve-

se realizar uma otimização no tamanho das regiões de forma a minimizar as regiões que se sobrepõem, ajustando os raios de cobertura dos objetos representativos ou, quando necessário, removendo nós da estrutura.

O problema da sobreposição de regiões é comum em *MAMs* e afeta diretamente seu desempenho. Um problema básico e inerente aos espaços métricos é que estes não possuem a noção de área, volume ou qualquer outra medição existente no espaço vetorial. Além disso, deve-se identificar as áreas que se sobrepõem, os nós aos quais pertencem estas áreas e quantos objetos estão nestas áreas. Tudo isso a fim de tentar minimizar as sobreposições e os objetos que se encontram nesta condição. Para resolver esta questão, *Traina* propôs uma abordagem diferente. Ao invés de medir o espaço da sobreposição, conta-se a quantidade de objetos presentes nas subárvores que são abrangidas por ambas as regiões, com isto é possível se calcular o *fat-factor* da estrutura que irá auxiliar na decisão da execução do algoritmo para minimizar as regiões comuns entre os nós [3].

Formalmente, pode-se definir o *fat-factor* da seguinte maneira: Seja *T*, uma árvore métrica com altura *H* e *N* nós, $N \neq 1$ e seja *M* o número de objetos, então o *fat-factor* da árvore métrica *T* será:

$$fat(T) = \frac{I_c - H * N}{N} \cdot \frac{1}{M - H}$$

Onde I_c representa o número total de nós acessados, necessários para responder uma consulta pontual para cada um dos *N* objetos armazenados na árvore. Através deste mecanismo, pode-se medir de forma eficiente o grau de sobreposição dos nós da *Slim-Tree*. Um *MAM* ideal requer que somente um nó seja visitado em cada nível para responder uma consulta pontual. Assim, o *fat-factor* é um valor que deve variar entre 0 e 1 ($0 \leq$ *fatfactor(T)* ≤ 1), quando seu valor passar de um determinado limite o algoritmo *slim-down* deverá ser executado a fim ajustar as sobreposições existentes na árvore.

O algoritmo *slim-down* percorre a árvore construída e ajusta os raios dos nós movendo objetos entre nós irmãos ou eliminando nós que estejam com a sua taxa de utilização baixa. O algoritmo basicamente visita cada nó e verifica se os objetos mais distantes em relação ao representativo do nó podem ser movidos para nós irmãos. Fazendo assim, o raio de cobertura do nó analisado é minimizado também a área que se sobrepõem. Depois desta verificação, o algoritmo analisa se o nó está vazio e, neste caso, elimina o nó. Isto garante que a *Slim-Tree* terá, de certa forma, uma estrutura com a menor quantia de regiões de sobreposição possível.

2.4.1 Junção em Dados Métricos

A junção por similaridade é uma primitiva importante em banco de dados. Dada sua importância, ela tem sido muito utilizada na melhoria de algoritmos de mineração de dados, do inglês *data minning algorithms*, em aplicações no domínio de imagens médicas, multimídia, séries temporais, detecção

de anomalias, conjuntos de proteínas, dados astrofísicos, entre outros.

A junção por similaridade combina dois conjuntos de dados complexos em um conjunto resposta contendo todos os pares de objetos semelhantes baseado em um predicado de junção que normalmente é dependente da métrica adotada para o domínio dos conjuntos envolvidos. A definição geral para a junção por similaridade [4] é a seguinte:

$$R \bowtie_{\theta_P} S = \{\langle r, s\rangle | \theta_P(r, s), r \in R, s \in S\}$$

Onde *R* e *S* são as relações envolvidas na junção, *r* e *s* são os atributos do predicado de junção e θ_P representa o predicado da junção por similaridade. Este predicado especifica as condições baseadas na similaridade que os pares $\langle a, b\rangle$ devem satisfazer para que sejam resposta na junção por similaridade. Existem, na literatura, vários predicados de junção por similaridade que são usados para a construção de operadores de junções por similaridade, alguns do predicados são: junção por abrangência, junção por vizinhos mais próximos, junção por proximidade e junção ao redor.

A Figura 2.6 ilustra um exemplo para os tipos de junção por similaridade mencionados, aplicados em dois conjuntos de dados numéricos. Para se entender melhor o uso dos predicados na construção de operadores que implementam a junção por similaridade, os quatro exemplos anteriores são descritos com mais detalhes a seguir.

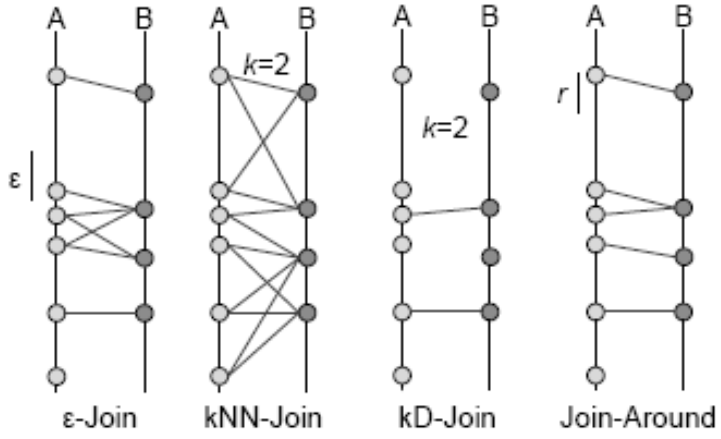

Figura 2.6: Tipos de junção por similaridade [4].

Muitos trabalhos têm sido desenvolvidos a fim de propor construções de junção no espaço métricos. Várias linhas de pesquisas são encontradas na literatura para resolver o problema de junção em espaços métricos. Dentre estas linhas, pode ser citada soluções que não utilizam estruturas para indexar os conjuntos de dados envolvidos. Jacox [16] apresenta uma comparação entre três dos algoritmos mais relevantes para este tipo de abordagem. Dentro desta abordagem existe o algoritmo *Epsilon Grid Order* (*EGO*). O Algoritmo *EGO* [22] se baseia em uma ordenação particular do conjunto de dados. O *EGO* utiliza uma grade de tamanho ε sobre o espaço e usa um planejador eficiente

para leitura dos blocos desta grade a fim de minimizar operações de I/O. O algoritmo GESS (*Generic External Space Sweep*) [23] cria hiper-quadrados centrados em cada ponto de dado com os lados de tamanho ε e combinam estes hiper-quadrados usando uma junção espacial sobre os retângulos. Por fim, o algoritmo *Quick Join* [16] particiona recursivamente os dados até os subconjuntos serem suficientemente pequenos de forma a permitir o processamento eficiente usando o algoritmo de junção por laço aninhado. O algoritmo cria chamadas recursivas para processar cada partição, e chamadas recursivas separadas para processar as janelas ao redor dos limites de cada partição.

Outros operadores de junção em espaço métrico utilizam estruturas de dados métricas para a indexação dos conjuntos de dados envolvidos na junção. Entre os trabalhos que utilizam estruturas baseadas em regiões circulares definidas por um raio de cobertura, *Paredes* e *Reyes* [24] propõem uma nova estrutura de indexação denominada lista de agrupamentos individuais (*List of Twin Clusters* ou *LTC*). O trabalho se baseia na proposta de Chávez [25] para construir uma estrutura fundamentada em listas e regiões de cobertura denominada agrupamentos as quais indexam os conjuntos envolvidos na junção. A estrutura cria duas listas de agrupamentos individuais que são usadas para consultas por abrangência. Existem ainda outras estruturas que dão suporte e são: uma matriz com as distâncias calculadas entre os objetos de um conjunto em relação ao outro e quatro vetores que armazenam o identificador do agrupamento e a distância máxima e mínima para cada objeto, a partir de um conjunto em direção aos centros do outro conjunto. A junção é resolvida, então, executando uma consulta por abrangência para os objetos a partir de um dos conjuntos recuperando os objetos relevantes no outro conjunto. Esta abordagem é muito boa, pois trabalha com uma estrutura baseada em regiões de cobertura, porém, seu grande problema é a necessidade de memória principal para a manutenção da matriz de distâncias e o fato de produzir, em determinadas situações, objetos que não são indexados, tendo que gerenciá-los.

Uma outra proposta envolvendo a construção de operadores de junção por similaridade que usa indexação através de um método de acesso métrico é proposto em [4]. No trabalho, os autores apresentam um estudo completo sobre predicados de junção por similaridade para a construção de operadores de junção por similaridade, além de propor um novo tipo de predicado denominado *Join-Around*. Basicamente, o *Join-Around* combina algumas propriedades de junção por abrangência (ε-join) e junção pelos k-vizinhos mais próximos fazendo com que todos os pares mais próximos de ambas as relações sejam encontrados e somente os pares separados por uma distância de até *r* são consideradas soluções onde $r = \frac{MD}{2}$, representa o *raio máximo* e *MD* representa o *diâmetro máximo* para a consulta.

2.4.2 Algoritmo de Junção por Abrangência (*Range Distance Join* - RDJ)

O *RDJ* recebe como parâmetros de entrada dois conjuntos de dados *R* e *S* pertencentes a um mesmo domínio e uma distância máxima ε. O predicado de junção busca todos os pares de objetos que estejam à distância ε uns dos outros ou $\theta_\varepsilon \equiv dist(a,b) \leq \varepsilon$. Formalmente, a *RDJ* é definida da seguinte forma:

$$R \overset{\bowtie}{_\varepsilon} S := \{(r_i, s_j) \in R \times S : dist(r_i, s_j) \leq \varepsilon\} \tag{2.1}$$

Algoritmo 2.12 rangeDistanceJoin(R,S,raio)

```
1: para cada tupla r ∈ R faça
2:    para cada tupla s ∈ S faça
3:       se distancia(r, s) ≤ raio então
4:          adiciona par ⟨r, s⟩ na lista de resultados
```

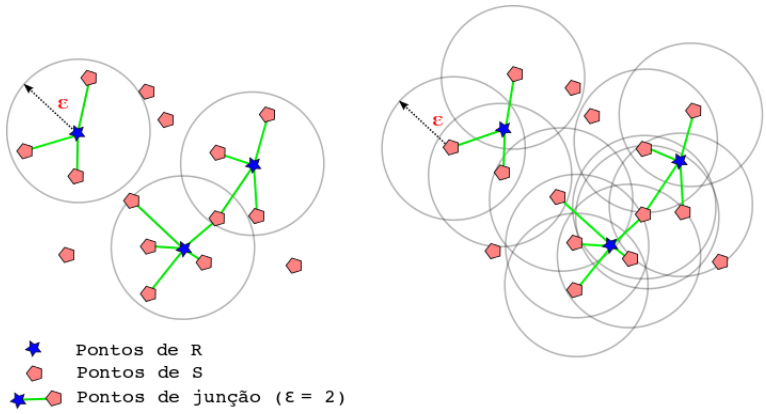

Figura 2.7: Comutatividade da junção *RDJ* desde que se considere o mesmo valor de ε.

O Algoritmo 2.12 exemplifica como a junção *RDJ* pode ser implementada com o uso de *loops* aninhados. Como esta operação é comutativa, ou seja, a ordem das relações não altera o resultado dos pares encontrados, desde que se considere o mesmo raio, pode-se usar a relação com a menor cardinalidade no laço externo a fim de melhorar o seu desempenho, a Figura 2.7 mostra um exemplo da comutatividade da junção *RDJ*.

2.4.3 Algoritmo de Junção por Vizinhos mais Próximos (*k-Nearest Neighbor Join* - kNNJ)

O *kNNJ* recebe como parâmetro de entrada dois conjuntos de dados *R* e *S* pertencentes a um mesmo domínio e uma quantidade *k* qualquer. O predicado de junção busca os *k* pares de objetos mais próximos do conjunto *R* para cada objeto do conjunto *S* ou $\theta_{kNN} \equiv b$ é um *k-vizinho mais próximo* de *a*. Formalmente, a *kNNJ* é definida da seguinte forma:

$$R \underset{K-nn}{\ltimes} S \tag{2.2}$$

$$\forall (r,s) \in R \underset{K-nn}{\ltimes} S, \forall (r,s') \in R \ltimes S \vee R \underset{K-nn}{\ltimes} S : dist(r,s) < dist(r,s') \tag{2.3}$$

Onde (2.2) é o menor subconjunto de $R \times S$ que contém, para cada ponto de *R*, no mínimo *k* pontos de *S*, para os quais a condição (2.3) se mantém verdadeira. Caso haja empates entre objetos na maior distância obtida, é comum duas abordagens, a primeira limita os objetos aos *k* elementos que satisfazem a regra ou cria-se uma lista de empates.

Algoritmo 2.13 kNearestNeighborJoin(R,S,k,empate)

```
1:  cria uma lista de resultados result com o tamanho de Rxk
2:  cria uma lista de resultados auxiliar resultAux com o tamanho de k
3:  para cada tupla r_i ∈ R faça
4:     para cada tupla s_j ∈ S faça
5:        se tamanhoresultAux ≤ k então
6:           adiciona o par ⟨r_i, s_j⟩ em resultAux
7:        senão
8:           se distancia(r_i, s_j) ≤ maior distância ∈ resultAux então
9:              adiciona o par ⟨r_i, s_j⟩ em resultAux
10:             remove de resultAux objetos excedentes à k verificando empate
11:    adiciona resultAux em result
```

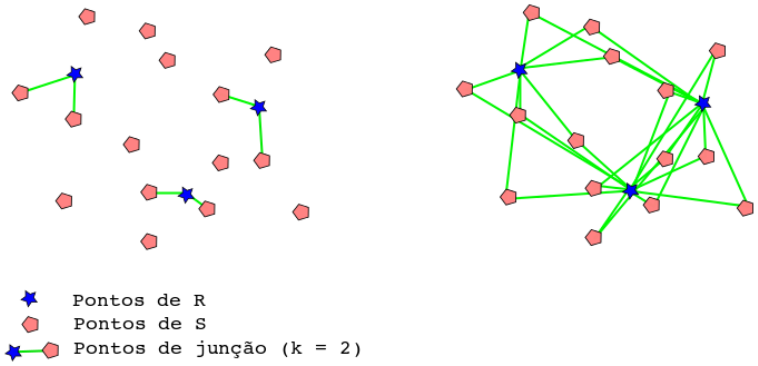

Figura 2.8: Não comutatividade da junção *kNNJ*.

O Algoritmo 2.13 exemplifica como a junção *kNNJ* pode ser implementada com o uso de *loops* aninhados. Como esta operação não é comutativa, ou seja, a ordem das relações altera o resultado dos pares encontrados, a melhoria no Algoritmo 2.12, usando a relação com a menor cardinalidade no laço externo não irá melhorar o seu desempenho. A Figura 2.8 mostra um exemplo da não comutatividade da junção *kNNJ*.

2.4.4 Algoritmo de Junção por Proximidade (*k-Distance Join* - kDJ)

O *kDJ* ou como também é chamada (*k-Closest Neighbor Join*) recebe como parâmetro de entrada dois conjuntos de dados *R* e *S* pertencentes a um mesmo domínio. O predicado de junção busca todos os k pares de $R \times S$ que possuem a menor distância entre os objetos de cada par ou $\theta_{kD} \equiv \langle a, b \rangle$ é um de todos os *k-closest pair*. Formalmente, o *kDJ* é definido da seguinte forma:

$$R \underset{k-CN}{\bowtie} S \tag{2.4}$$

$$\forall(r,s) \in R \underset{K-CN}{\bowtie} S, \forall(r',s') \in R \bowtie S \vee R \underset{K-CN}{\bowtie} S : dist(r,s) < dist(r',s') \tag{2.5}$$

Onde (2.4) é o menor subconjunto de $R \times S$ que contém, no mínimo, *k* pares de pontos de *S*, para os quais a condição (2.5) se mantém verdadeira.

Algoritmo 2.14 kDistanceJoin(R,S,k,empate)

```
1: cria uma lista de resultados result com o tamanho de k
2: para cada tupla r_i ∈ R faça
3:    para cada tupla s_j ∈ S faça
4:       se tamanhoderesult < k então
5:          adiciona o par ⟨r_i, s_j⟩ em resultAux
6:       senão
7:          se distancia(r_i, s_j) ≤ maior distância ∈ result então
8:             adiciona o par ⟨r_i, s_j⟩ em result
9:             remove de result objetos excedentes à k verificando empate
```

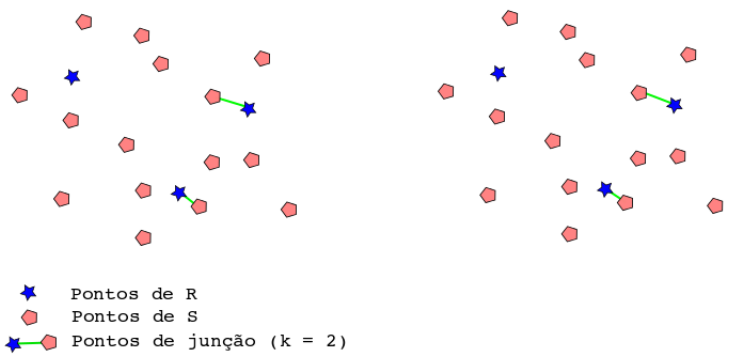

Figura 2.9: Comutatividade da junção *kDJ* desde que se considere $\langle r,s\rangle = \langle s,r\rangle$.

O Algoritmo 2.14 exemplifica como a junção *kDJ* pode ser implementada com o uso de *loops* aninhado. Como esta operação é comutativa, ou seja, a ordem das relações não altera o resultado dos pares encontrados, desde que se considere que os pares $\langle r,s\rangle$ e $\langle s,r\rangle$ sejam os mesmos pares, pode-se usar a relação com a menor cardinalidade no laço externo a fim de melhorar o seu desempenho, a Figura 2.9 mostra um exemplo da comutatividade da junção *kDJ*.

2.4.5 Algoritmo de Junção ao Redor - (*Join Around*)

O *JA* recebe como parâmetro de entrada dois conjuntos de dados *R* e *S* pertencentes a um mesmo domínio. O predicado de junção recupera todos os *k* pares de $R \times S$ que possuem a menor distância entre os objetos de cada par e também que estejam a uma distância máxima ε uns dos outros ou $\theta_{A,MD=2r} \equiv b$ é um dos vizinhos mais próximos de *a* e $d(a,b) \leq r$. Formalmente, a *JA* é definida da seguinte forma:

$$R \underset{A,MD=2r}{\bowtie} S \tag{2.6}$$

$$\forall (r,s) \in R \underset{A,MD=2r}{\bowtie} S, \forall (r',s') \in R \times S \vee R \underset{A,MD=2r}{\bowtie} S : dist(r,s) < dist(r',s') \wedge dist(r,s) \leq r \quad (2.7)$$

Onde (2.6) é o menor subconjunto de $R \times S$ que contém, para cada ponto de *R*, no mínimo *k* pares de pontos de *S* e que estejam a no máximo *r* de distância, para os quais a condição (2.7) se mantém verdadeira.

Algoritmo 2.15 joinAround(R,S,k,empate,raio)

```
1:  cria uma lista de resultados result com o tamanho de Rxk
2:  cria uma lista de resultados auxiliar resultAux com o tamanho de k
3:  para cada tupla r_i ∈ R faça
4:     para cada tupla s_j ∈ S faça
5:        se tamanhoresultAux ≤ k então
6:           adiciona o par ⟨r_i, s_j⟩ em resultAux
7:        senão
8:           se distancia(r_i, s_j) ≤ maior distância ∈ resultAux então
9:              adiciona o par ⟨r_i, s_j⟩ em resultAux
10:             remove de resultAux objetos excedentes à k verificando empate
11: para cada tupla res_n ∈ resultAux faça
12:    se distancia(dopar⟨r_n, s_n⟩ ∈ res_n) ≤ raio então
13:       adiciona res_n em result
```

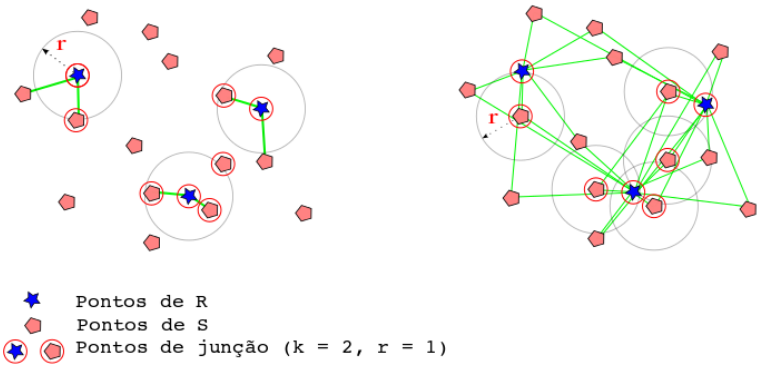

Figura 2.10: Comutatividade da junção *JA* desde que se considere $\langle r,s \rangle = \langle s,r \rangle$ e o mesmo raio *r*.

O Algoritmo 2.15 exemplifica como a junção *JA* pode ser implementada com o uso de *loops* aninhados. Apesar desta operação usar propriedades da *kNNJ*, ela também é comutativa por causa das propriedades da *RDJ*, ou seja, a ordem das relações não altera o resultado dos pares encontrados, desde que se considere que os pares $\langle r,s \rangle$ e $\langle s,r \rangle$ sejam os mesmos pares e também considerando o mesmo raio *r*, pode-se usar a relação com a menor cardinalidade no laço externo a fim de melhorar o seu desempenho. A Figura 2.10 mostra um exemplo da comutatividade da junção *kDJ*.

2.5 Considerações Finais

Este capítulo apresentou os conceitos e definições que envolvem a operação de junção em dados de ordem total. Em seguida, foi apresentado o estado da arte da junção espacial e junção por similaridade. As discussões relacionadas a junção mostram que existem várias linhas de pesquisas, algumas

delas envolvendo indexação dos dados do domínio estudado e outras sem a indexação dos dados do domínio.

Muitos trabalhos na área de indexação e recuperação de trajetória possuem propostas que se assemelham a este trabalho. Ke Deng [26] apresentou o seguinte problema: “Deseja-se encontrar todos os postos de gasolina mais próximos, ao longo de um segmento de trajetória específico entre dois pontos”. Esta consulta pode ser respondida usando-se um tipo de consulta por trajetória chamada *P-Query*. A *P-Query* procura por pontos de interesse (POI - *Points of Interest*) que satisfazem um relacionamento espaço temporal para um segmento (s) que representa uma trajetória específica, ou busca trajetórias que satisfaçam um relacionamento espaço temporal para um ponto específico em um dado conjunto de pontos.

Embora existam semelhanças no tipo de questão a ser respondida, este trabalho não se propõe a ser uma solução para problemas que envolvam trajetória. Apesar de ter rotas como objeto auxiliar durante a consulta, este trabalho é, essencialmente, uma proposta para um novo operador de junção em banco de dados métrico.

O capítulo seguinte define um novo operador chamado *Junção Canalizada*, apresenta definições, técnicas de implementação e geometrias envolvidas em sua aplicação.

Capítulo
3
Junção Canalizada

3.1 Considerações Iniciais

Neste capítulo será apresentado um novo operador de junção chamado Junção Canalizada "*JC*" que responde a consultas do tipo: "Quais são as cidades que estão até 16 km da rota entre Pouso Alegre / MG e Itajubá / MG". A figura 3.1 ilustra essa consulta sendo que: a rota é representada pelas cidades Pouso Alegre – Santa Rita do Sapucaí – Piranguinho – Itajubá; a distância de 16 km para essa rota forma o canal que cobre as cidades que serão resposta para essa consulta: Pouso do Campo, São Sebastião da Bela Vista, Cruz do Bento João, Cachoeira de Minas, São José do Alegre e Wenceslau Brás. No conjunto de pontos que formam a base de dados existem também pontos do domínio que não fazem parte da resposta por não estarem dentro do canal de 1 km como: Brasópolis, Pedralva e Maria da Fé.

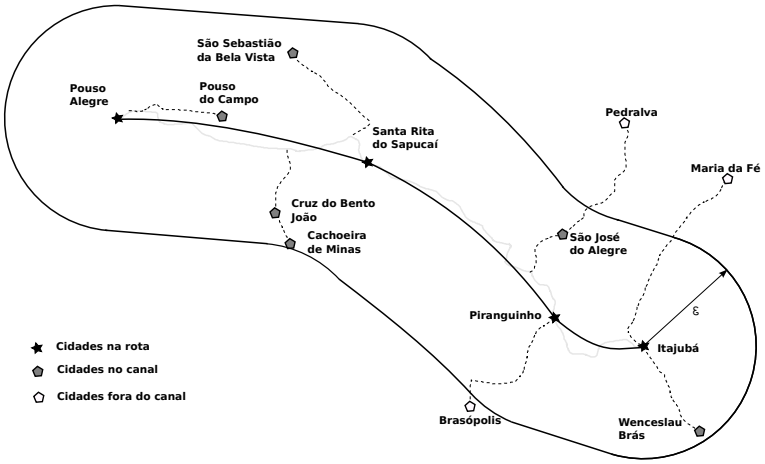

Figura 3.1: Dimensões do canal *JC*.

É importante mencionar também que outros tipos de problemas que envolvam a noção de rota, como por exemplo: caminho, linha, trajetória, duto, curso, entre outras, podem ser beneficiadas pela junção canalizada, desde que atendam as restrições da seção 3.2. Como exemplo de problemas que

podem ser solucionados pela *JC*, sugere-se:

- "Dado um caminho por onde será construído um oleoduto, mostrar os pontos de maior incidência de queimadas a menos de 5 km deste caminho."
- "Dado uma linha de pesquisa, encontre as pesquisas que estejam a menos de 10 pontos de relevância."
- "Dado uma trajetória padrão para uma dada carreira, encontre os profissionais cujas carreiras estejam a menos de 5 unidades de distância."
- "Dado uma linha de consumo para determinados perfis de clientes, encontre os clientes que possuam perfis a menos de 100 unidades de distância desta linha."

Na seção 3.2 serão apresentadas as definições para a Junção Canalizada (*JC*) aplicadas a dados sujeitos a geometria euclidiana e geometria esférica.

3.2 Definições

Da mesma forma que a junção por abrangência apresentada na seção 2.4.2, a *Junção Canalizada* também recebe como parâmetros de entrada dois conjuntos de dados *R* e *S* pertencentes a um mesmo domínio e uma distância máxima ε. A diferença é que um dos conjuntos representa uma rota de pontos que forma um grafo conexo acíclico de grau máximo dois.

Definindo que o primeiro conjunto é formado por *n* pontos distintos do domínio, como por exemplo, as latitudes e longitudes das cidades brasileiras e o segundo conjunto é formado pelo conjunto de pontos que formam uma rota, por exemplo, as latitudes e longitudes da rota Pouso Alegre – Santa Rita do Sapucaí – Piranguinho – Itajubá. A junção deve encontrar todos os pontos do domínio (primeiro conjunto) que estejam a uma distância máxima ε da rota de pontos (segundo conjunto).

A estratégia para responder essa operação de junção é verificar se o ponto no domínio atende uma das duas situações de qualificação:

1. Se a distância até um dos pontos da rota é menor que a distância máxima ε ou;
2. Se a menor distância para a rota é menor que a distância máxima ε.

Para ilustrar essas situações de qualificação observe a figura 3.2 que tem: a rota Piranguinho – Itajubá representados por estrelas; os pontos do domínio a, b, c representados por pentágonos; e o raio de canal que é de 16 km.

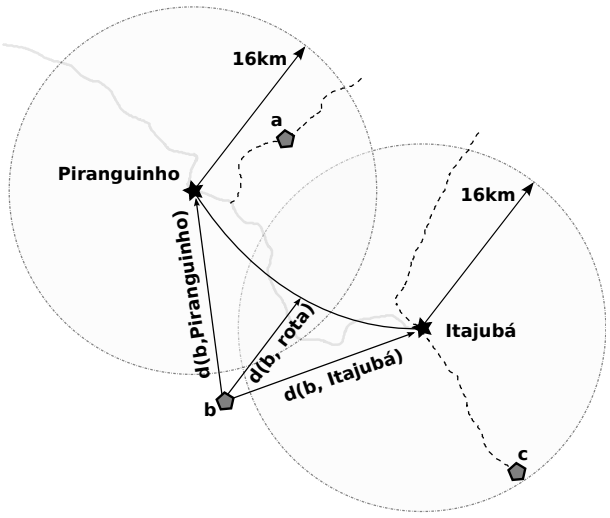

Figura 3.2: Distância de um ponto a para a rota.

Na primeira estratégia de qualificação existem cidades do domínio (pentágonos) que estarão a menos do que 16 km de pelo menos uma cidade da rota (estrela), logo esta cidade do domínio fará parte do conjunto resposta. A figura 3.2 ilustra um exemplo desta situação onde o ponto *a* tem distância menor que 16km em relação ao ponto Piranguinho na rota e o ponto *c* tem distância menor do que 16 km em relação ao ponto Itajubá na rota.

No entanto, existem cidades do domínio, como o ponto *b*, cuja distância para as cidades Piranguinho – Itajubá na rota é maior do que 16 km, mas a sua distância para a rota formada entre os pontos Piranguinho – Itajubá é inferior a 16 km. Neste caso, é necessário utilizar a geometria sobre a qual estão sujeitos os pontos para medir a menor distância entre o ponto do domínio e a rota existente entre os outros dois pontos.

3.3 Geometria

A geometria é um ramo da matemática em que são tratadas as propriedades de figuras no espaço [27]. *Euclides* fundamentou sua teoria em vinte e três definições relacionadas aos elementos básicos. Uma definição importante é a vigésima terceira que define retas paralelas e será apresentada a seguir:

Definição 3.3.1. *"Retas paralelas são retas que iniciam no mesmo plano e continuam indefinidamente em ambas as direções sem se encontrarem em qualquer das direções."*

Essa definição identifica quando uma geometria é euclidiana e quando uma geometria não é euclidiana. Desta forma, pode-se usar os elementos e relações da geometria euclidiana para solucionar problemas que possam ser representados no plano.

Nos casos de geometria não euclidiana deve-se buscar elementos e relações próprias para este domínio. Um exemplo de geometria não euclidiana é a geometria esférica.

O uso de geometria para encontrar a distância do ponto até a rota será detalhado na próxima seção.

3.3.1 Geometria no Espaço Euclidiano

O elemento de estudo deste trabalho dentro da geometria euclidiana é o triângulo e as relações existentes entre seus lados e ângulos.

Dessa forma, dado dois pontos consecutivos do domínio na rota r_1 e r_2, tem-se uma reta "**a**" medindo a distância entre r_1 e r_2. É possível dividir o espaço em três regiões traçando duas retas paralelas entre si e perpendicular a reta **a**, uma passando pelo ponto r_1 e outra passando por r_2. A figura 3.3 ilustra as regiões obtidas. O canal que qualifica pontos do domínio é formado pela distância máxima ε da reta **a**.

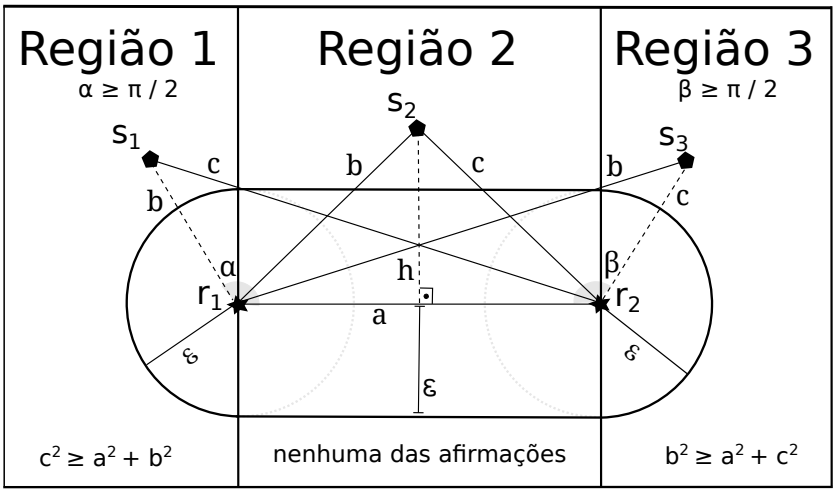

Figura 3.3: Regiões definidas sobre o plano.

Para determinar que um ponto qualquer s_1 do domínio se encontra na região 1, deve haver um ângulo obtuso entre as retas formadas a partir das distâncias: de r_1 para r_2 (ambos pontos da rota); e de s_1 (ponto do domínio) para r_1 (ponto na rota). Da mesma maneira, para que um ponto qualquer s_3 do domínio se encontre na região 3 deve haver um ângulo obtuso entre as retas formadas a partir das distâncias: de r_1 para r_2 (ambos pontos da rota); e de r_2(ponto na rota) para s_3 (ponto do domínio).

Assim em qualquer uma das regiões 1, 2 e 3, tem-se que:

- a é a distância entre dois pontos consecutivos r_1 e r_2 da rota;
- b é a distância do ponto *s* do domínio para o ponto r_1 da rota e;

- c é a distância do ponto *s* do domínio para o ponto r_2 da rota.

Pela lei dos cossenos pode-se determinar que um ângulo é obtuso, em um triângulo, quando o quadrado do cateto oposto ao ângulo é maior que a soma dos quadrados dos catetos adjacentes ao angulo. Assim, é possível determinar que um ponto *s* do domínio está em uma das regiões, da seguinte forma:

- se $c^2 \geq a^2 + b^2$ então o ponto *s* do domínio está na *Região 1*;
- se $b^2 \geq a^2 + c^2$ então o ponto *s* do domínio está na *Região 3* e;
- se nenhuma das condições anteriores forem verdadeiras o ponto *s* tem que estar na *Região 2* da figura.

Para que o ponto *s* do domínio seja resposta da consulta na região 1, sua distância para o ponto da rota r_1 deve ser menor que a distância máxima ε do canal. Da mesma forma, para que o ponto *s* do domínio seja resposta da consulta na região 3, sua distância para o ponto da rota r_2 deve ser menor que a distância máxima ε.

Por fim, para determinar que o ponto *s* do domínio seja resposta da consulta na região 2, deve-se usar relações geométricas euclidianas para determinar a menor distância do ponto do domínio para a reta formada pela distância entre os pontos r_1 e r_2 na rota. Pode-se ver pela figura 3.3, que a menor distância procurada é exatamente a altura do triângulo que tem como base a reta formada pela distância entre os pontos r_1 e r_2 (ambos da rota).

Para encontrar o valor dessa altura calcula-se inicialmente o perímetro do triângulo usando o *Teorema de Heron* da seguinte forma:

$$P = (a + b + c)/2$$

Substituindo os lados do triângulo pela função de distância $d()$ entre dois pontos no espaço euclidiano, tem-se que:

$$P = (d(r_1, r_2) + d(r_1, s) + d(s, r_2))/2$$

Em seguida, calcula-se a área do triângulo também usando o *Teorema de Heron*, sendo:

$$A = \sqrt{P(P-a)(P-b)(P-c)}$$

Substituindo os lados do triângulo da mesma maneira tem-se que:

$$A = \sqrt{P(P-d(r_1, r_2))(P-d(r_1, s))(P-d(s, r_2))}$$

Finalmente, substituindo na fórmula tradicional de cálculo da área do triângulo, base multiplicada pela altura dividido por dois, obtem-se a menor distância buscada (altura) dada por:

$$h = \frac{2 \cdot A}{d(r_1, r_2)}$$

Assim, para que um ponto *s* qualquer do domínio seja resposta da consulta na região 2, o valor de *h* definido acima deve ser menor que a distância máxima ε.

3.3.2 Geometria no Espaço Esférico

O espaço esférico é o primeiro espaço não euclidiano estudado na literatura matemática. Com exceção das discussões sobre retas paralelas, todas as outras proposições e postulados podem ser verificados nos demais espaços. Sendo assim, um espaço onde a teoria das paralelas não seja verdadeira é considerado um espaço não euclidiano [28].

Dados dois pontos consecutivos do domínio da rota r_1 e r_2, tem-se um arco "**a**" medindo a distância entre esses dois pontos. É possível dividir o espaço em três regiões traçando dois arcos perpendiculares ao arco **a**, um passando pelo ponto r_1 e outro passando por r_2. A figura 3.4 ilustra esta situação e apresenta as regiões e o triângulo esférico que são construídos sobre a superfície da esfera. O canal que qualifica pontos do domínio é formado pela distância máxima ε do arco **a**.

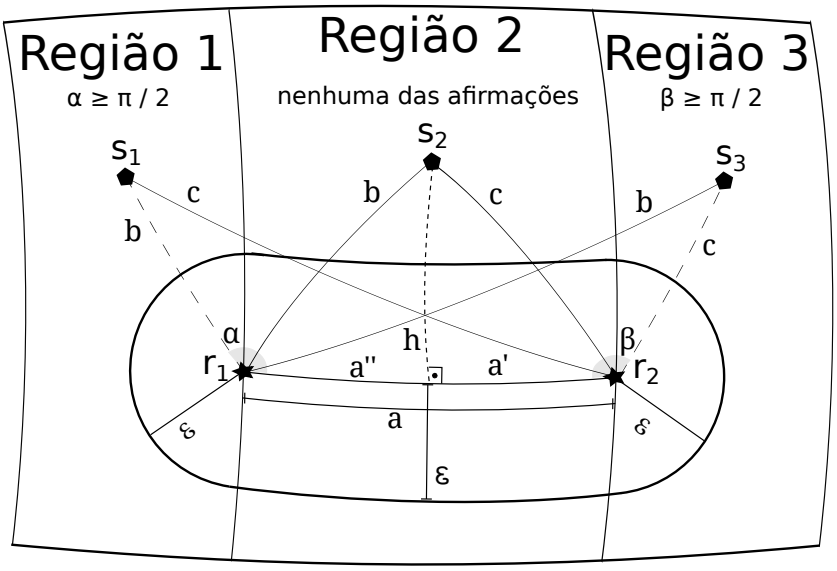

Figura 3.4: Regiões definidas sobre a esfera.

Para determinar em qual das três regiões encontra-se um ponto *s* qualquer do domínio, é necessário calcular os valores dos ângulos α e β mostrados na figura 3.4. Pode-se deduzir o valor dos ângulos usando a lei esférica dos cossenos também chamada de *regra de cosseno para lados* [29], onde: *a*, *b*, e *c* são os lados do triângulo esférico. A *regra de cosseno para lados* é apresentada a seguir:

$$\cos(c) = \cos(a)\cos(b) + \sin(a)\sin(b)\cos(\alpha),$$

Substituindo os lados do triângulo esférico pela função de distância $d()$ entre dois pontos no espaço esférico, tem-se os ângulos:

$$\alpha = \arccos(c^2 - a^2 - b^2 + 2 \times a \times b)$$
$$\beta = \arccos(b^2 - a^2 - c^2 + 2 \times a \times c)$$

A função para calcular a distância entre dois pontos na esfera é dada por:

$$d(i,j) = raio \times \arccos(\sin(i.lat) \times \sin(j.lat) + \cos(i.lat) \times \cos(l.lat) \times \cos(j.long - i.long))$$

Para determinar que um ponto s_1 qualquer do domínio encontra-se na região 1, deve haver um ângulo obtuso entre os arcos formados a partir das distâncias: de r_1 para r_2 (ambos pontos da rota); e de s_1 (ponto do domínio) para r_1 (ponto na rota). Nesta situação, o ponto s_1 do domínio será resposta da consulta, se sua distância para o ponto da rota r_1 for menor do que a distância máxima ε do canal.

Da mesma maneira, para que um ponto s_3 qualquer do domínio se encontre na região 3, deve haver um ângulo obtuso entre os arcos formados a partir das distâncias: de r_1 para r_2 (ambos pontos da rota); e de r_2(ponto na rota) para s_3 (ponto do domínio). Nesta situação, o ponto s_3 do domínio será resposta da consulta se sua distância para o ponto da rota r_2 for menor do que a distância máxima ε.

Por fim, para determinar que o ponto *s* qualquer do domínio é resposta da consulta na região 2, deve-se usar as relações geométricas esféricas para determinar a menor distância do ponto do domínio para o arco formado pela distância entre os pontos r_1 e r_2 da rota. Como pode ser visto na figura 3.4, a menor distância procurada é representada pela altura do triângulo esférico que tem como sua base o arco formado pela distância dos pontos r_1 e r_2 (ambos da rota).

Pelo teorema esférico dos senos tem-se para o triângulo esférico *h, c, a'*, a seguinte definição:

$$\frac{sen(h)}{sen(\widehat{H})} = \frac{sen(c)}{sen(\widehat{C})} = \frac{sen(a')}{sen(\widehat{A'})}$$
$$\frac{sen(h)}{sen(\widehat{H})} = \frac{sen(c)}{sen(90)}$$
$$\frac{sen(h)}{sen(\widehat{H})} = sen(c)$$
$$h = arcsen(sen(c) \times sen(\widehat{H}))$$

Substituindo *c* pela distância do ponto s_2 no domínio para o ponto r_2 na rota obtem-se:

$$h = arcsen(sen(d(s_2, r_2)) \times sen(\widehat{H}))$$

A figura 3.4 mostra o triângulo esférico *h, c, a'* inscrito no triângulo esférico *a, b, c* pelos ângulos $\widehat{H}$ (oposto ao lado *h* do triângulo esférico *h, c, a'*) e $\widehat{B}$ (oposto ao lado *b* do triângulo esférico *a, b, c*). Sendo assim, $\widehat{H}$ igual a $\widehat{B}$.

É possível determinar o ângulo $\widehat{B}$, usando a regra de cosseno para lados, em que:

$$\cos(b) = \cos(c)\cos(a) + \sin(c)\sin(a)\cos(\widehat{B}),$$

Substituindo os lados do triângulo esférico pela função de distância $d()$ entre dois pontos no espaço esférico, tem-se que:

$$\cos(d(r_1, s_2)) = \cos(d(s_2, r_2))\cos(d(r_1, r_2)) + \sin(d(s_2, r_2))\sin(d(r_1, r_2))\cos(\widehat{B})$$
$$\frac{\cos(d(r_1, s_2)) - \cos(d(s_2, r_2))\cos(d(r_1, r_2))}{\sin(d(s_2, r_2))\sin(d(r_1, r_2))} = \cos(\widehat{B})$$
$$\widehat{H} = \widehat{B} = \arccos(\frac{\cos(d(r_1, s_2)) - \cos(d(s_2, r_2))\cos(d(r_1, r_2))}{\sin(d(s_2, r_2))\sin(d(r_1, r_2))})$$

3.3.3 Junção Canalizada usando Processamento Sequencial

O algoritmo de junção canalizada usando processamento sequencial, percorre todos os elementos s_i do domínio S. Para cada elemento do domínio deve-se verificar se algum par consecutivo de pontos na rota (r_{j-1}, r_j) consegue qualificar o ponto s_i como resposta baseado em uma das 3 regiões.

Algoritmo 3.1 JCSequencial(pontos do domínio S, pontos da rota R, ε)

```
1:  para cada ponto s_i no domínio S faça
2:    para cada ponto r_j no domínio R, sendo j iniciado com 2 faça
3:      determinar regiao n, sendo os pontos da rota r_{j-1} e r_j
4:      se n = 1 então
5:        // REGIAO 1
6:        se d(s_i, r_{j-1}) ≤ ε então
7:          adiciona s_i a resposta
8:      senão
9:        se n = 3 então
10:         // REGIAO 3
11:         se d(s_i, r_j) ≤ ε então
12:           adiciona s_i a resposta
13:       senão
14:         // REGIAO 2
15:         dist ← menor distância entre o ponto s_i e a rota (r_{j-1}, r_j)
16:         se dist ≤ ε então
17:           adiciona s_i a resposta
```

Como pode ser visto no algoritmo 3.1, existem dois laços aninhados, sendo: um para percorrer o conjunto de pontos do domínio S e outro para percorrer o conjunto de pontos da rota R seguida pelas condições de qualificação para a resposta. Os parâmetros para o algoritmo são os pontos do domínio S, os pontos consecutivos da rota R e a distância máxima do canal ε.

Na linha 10 do algoritmo 3.1 deve-se usar a geometria adequada ao domínio para calcular a menor distância entre o ponto s_i do domínio e a rota (r_{j-1}, r_j).

3.3.4 Junção Canalizada usando *Slim-Tree*

Para o funcionamento do algoritmo Junção Canalizada usando a estrutura métrica *Slim-Tree*, os pontos s_i do domínio S devem estar indexados nesta estrutura. O conjunto de pontos da rota deve estar em uma lista de elementos.

Algoritmo 3.2 *JCSlim(blocoSlim, ε, idxRota, distQualific)*

```
1: Localiza o objeto central da rota
2: para cada elemento objSlim_i ∈ blocoSlim faça
3:    dist ← d(objSlim_i, objRota_centro)
4:    se não poda pelo objeto da rota que gera menor cobertura entre todos pontos da rota então
5:       para cada elemento objRota_j ∈ rota, sendo j iniciado com idxRota faça
6:          se não poda pela distância entre o objSlim_i e seu representativo então
7:             determinar região n, sendo os pontos da rota objRota_{j-1} e objRota_j
8:             se blocoSlim = folha então
9:                // TRATAMENTO DE BLOCO FOLHA DA SLIM-TREE
10:               se n = 1 então
11:                  // REGIÃO 1
12:                  dist ← d(objSlim_i, objRota_{j-1})
13:                  se poda pela máxima cobertura do objRota_j entre todos pontos da rota então
14:                     interrompa
15:                  se dist ≤ ε então
16:                     adiciona objSlim_i a resposta
17:               senão
18:                  se n = 3 então
19:                     // REGIÃO 3
20:                     dist ← d(objSlim_i, objRota_j)
21:                     se poda pela máxima cobertura do objRota_j entre todos pontos da rota então
22:                        interrompa
23:                     se dist ≤ ε então
24:                        adiciona objSlim_i a resposta
25:                  senão
26:                     // REGIÃO 2
27:                     dist ← menor distância entre o ponto objSlim_i e a rota objRota_{j-1} − objRota_j
28:                     se poda pela máxima cobertura do objRota_j entre todos pontos da rota então
29:                        interrompa
30:                     se dist ≤ ε então
31:                        adiciona objSlim_i a resposta
32:            senão
33:               // TRATAMENTO DE BLOCO ÍNDICE DA SLIM-TREE
34:               se n = 1 então
35:                  // REGIÃO 1
36:                  dist ← d(objSlim_i, objRota_{j-1})
37:                  se poda pela máxima cobertura do objRota_j entre todos pontos da rota então
38:                     interrompa
39:                  se dist ≤ cobObjSlim_i + ε então
40:                     JCSlim( subBlocoSlim_i, ε, idxRota, dist )
41:               senão
42:                  se n = 3 então
43:                     // REGIÃO 3
44:                     dist ← d(objSlim_i, objRota_j)
45:                     se poda pela máxima cobertura do objRota_j entre todos pontos da rota então
46:                        interrompa
47:                     se dist ≤ cobObjSlim_i + ε então
48:                        JCSlim( subBlocoSlim_i, ε, idxRota, dist )
49:                  senão
50:                     // REGIÃO 2
51:                     dist ← menor distância entre o ponto objSlim_i e a rota objRota_{j-1} − objRota_j
52:                     se poda pela máxima cobertura do objRota_j entre todos pontos da rota então
53:                        interrompa
54:                     se dist ≤ cobObjSlim_i + ε então
55:                        JCSlim( subBlocoSlim_i, ε, idxRota, dist )
```

Como apresentado no algoritmo 3.2, existem dois laços aninhados sendo: um para percorrer o conjunto de objetos da *Slim-Tree* (pontos do domínio S) e outro para percorrer o conjunto de pontos da rota R seguido das 3 condições de poda para essa junção. Antes de iniciar o processamento dos laços, o algoritmo 3.2 localiza o objeto central da rota juntamente com a sua cobertura mínima necessário para processar a primeira poda do algoritmo.

As 3 condições de poda usando a estrutura *Slim-Tree* são:

1. poda pelo objeto da rota que gera menor cobertura entre todos pontos da rota (linha 3) – que permite a não visita aos nós na situação em que a distância do objeto $objSlim_i$ para o $objRota_{centro}$ é maior que a soma dos valores:
   - da cobertura máxima do $objRota_{centro}$ da rota;
   - da distância máxima da rota ε e;
   - somente em nós índices: da cobertura do objeto$_i$ da *Slim-Tree* ($cobObjSlim_i$).

2. poda pela distância entre o $objSlim_i$ e seu representativo (linha 5) – que permite a não visita aos nós usando a propriedade da desigualdade triangular na situação em que distância do objeto da rota r_j para o representativo do nó (já calculada na chamada anterior da recursão) é maior que a soma dos valores:
   - da distância do objeto $objSlim_i$ e seu representativo;
   - da distância máxima da rota ε e;
   - somente em nós índices: da cobertura do objeto$_i$ da *Slim-Tree* ($cobObjSlim_i$).

3. poda pela máxima cobertura do $objRota_j$ entre todos pontos da rota (linhas 12, 20, 27, 36, 44 e 51) – que permite a não visita aos nós na situação em que distância do objeto $objSlim_i$ para o $objRota_j$ é maior que a soma dos valores:
   - da cobertura máxima do $objRota_j$ da rota;
   - da distância máxima da rota ε e;
   - somente em nós índices: da cobertura do objeto$_i$ da *Slim-Tree* ($cobObjSlim_i$).

   A figura 3.5 ilustra um exemplo da primeira poda executada pelo algoritmo do operador de *Junção Canalizada*. Nesta poda, o objeto representado pela cidade de *Santa Rita do Sapucaí* gera a menor cobertura necessária para englobar todos os outros objetos presentes na rota, representados pelas outras cidades. A distância entre a cidade de *Santa Rita do Sapucaí* e a

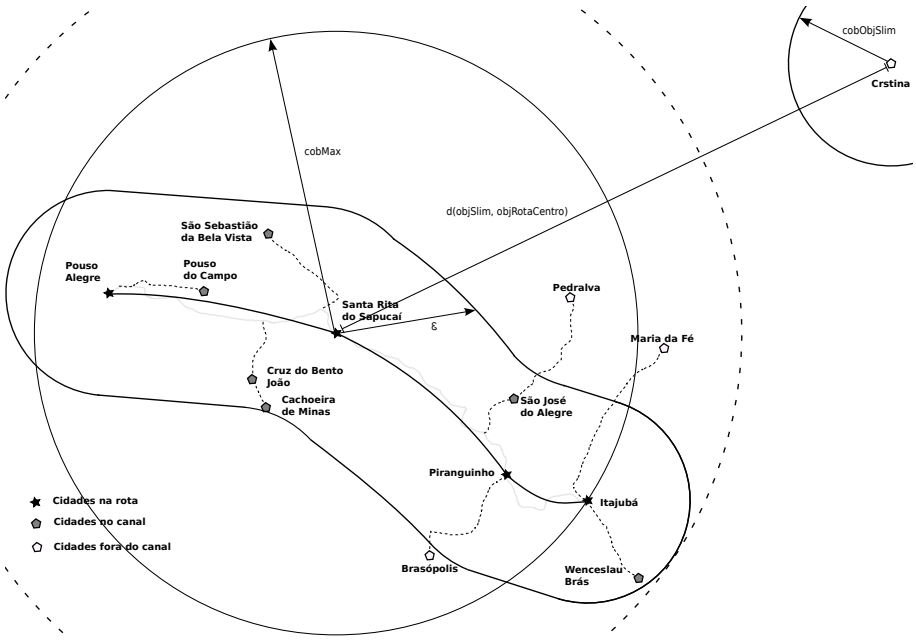

Figura 3.5: Poda em nó índice pelo objeto na rota que gera menor cobertura.

cidade de *Cristina* é maior do que a soma da cobertura máxima (*cobMax*) gerada pela cidade de *Santa Rita do Sapucaí* mais o tamanho do canal (ε) mais (no caso de nó índice) a cobertura (*cobObjSlim*) da cidade de *Cristina*. Dessa forma, o nó representado pela cidade de *Cristina* não precisa ser visitado.

A figura 3.6 ilustra as verificações que o algoritmo 3.2 faz para um nó índice da *Slim-Tree*. Como pode ser visto, os objetos índices que se qualificam na *Slim-Tree* foram s_{13}, s_{23} e s_{33}. Os objetos índices que foram podados pela segunda poda são: s_{12} pela região 1, s_{22} pela região 2, e s_{32} pela região 3.

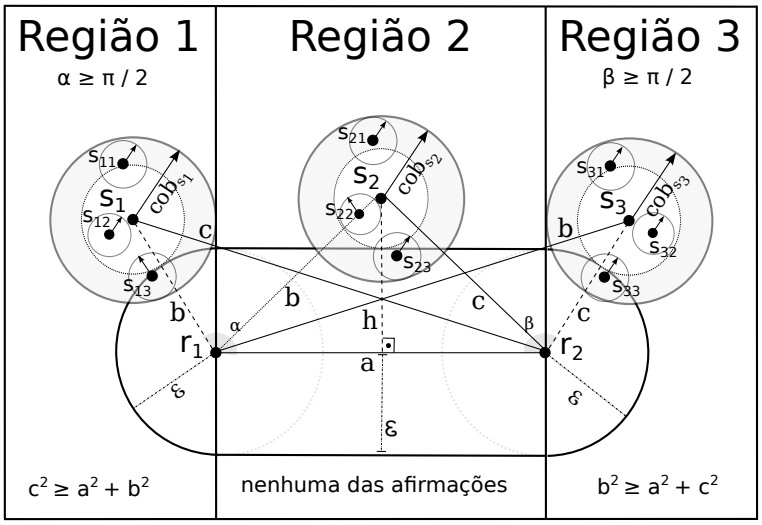

Figura 3.6: Poda em nó índice pela distância entre o objeto do nó e seu representativo.

A figura 3.7 ilustra as verificações que o algoritmo 3.2 faz para um nó índice da *Slim-Tree*. Como pode ser visto na figura 3.7, os objetos folhas que qualificaram-se como resposta são: s_{13} pela região

1, s_{23} pela região 2, e s_{33} pela região 3 entre os pontos da rota r_1 e r_2. Os objetos folhas que foram podados pela 2ª poda são: s_{12} pela região 1, s_{22} pela região 2, e s_{32} pela região 3.

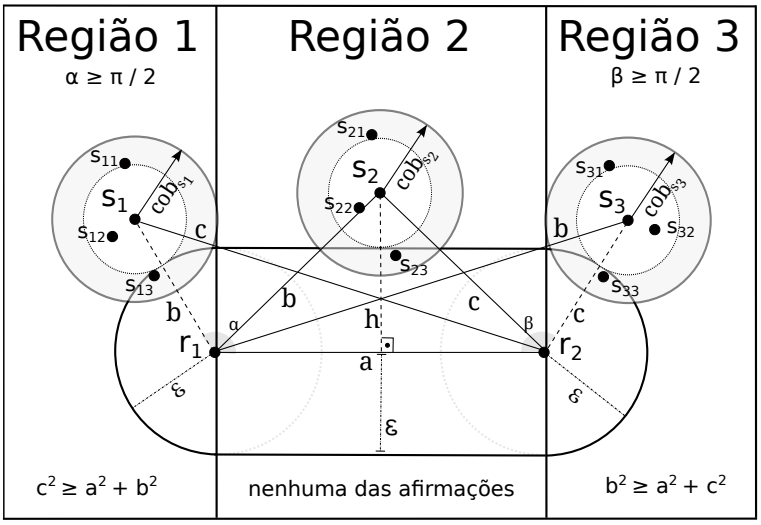

Figura 3.7: Poda em nó folha pela distância entre o objeto do nó e seu representativo.

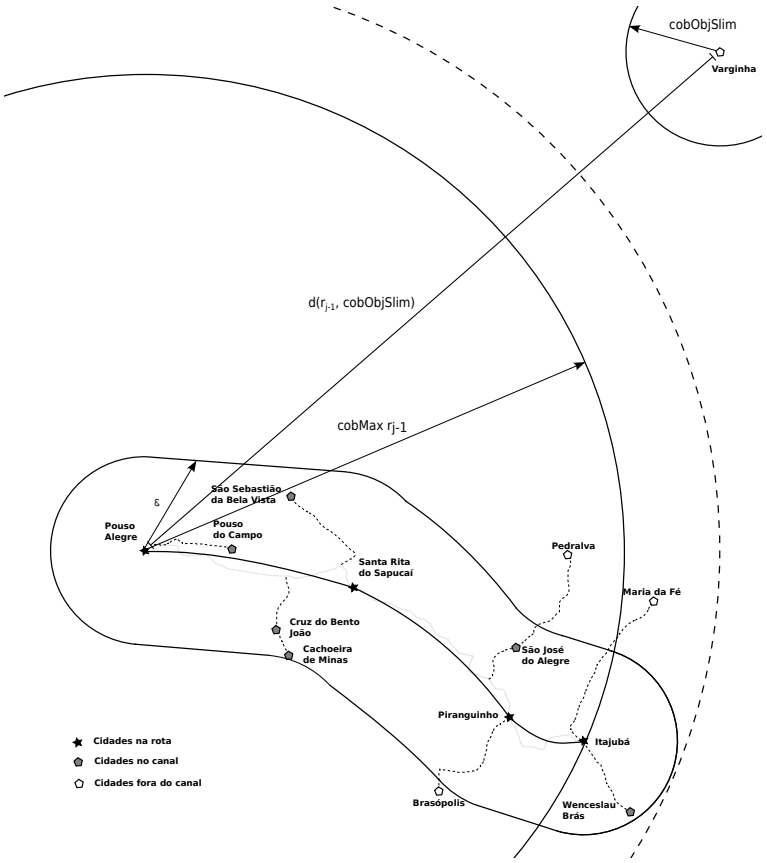

Figura 3.8: Poda pela máxima cobetura do *objRota* entre todos os pontos da rota.

Por fim, a figura 3.8 ilustra a poda executada pelo algoritmo do operador de *Junção Canalizada* usando a coberturá máxima gerada entre todos os pontos da rota. Nesta poda, o objeto representado pela cidade de *Pouso Alegre* gera a maior cobertura necessária para englobar todos os outros objetos presentes na rota, representados pelas outras cidades. A distância entre a cidade de *Pouso Alegre* e

a cidade de *Cristina*, é maior do que a soma da cobertura máxima (*cobMax*) gerada pela cidade de *Pouso Alegre* mais o tamanho do canal (ε) mais (no caso de nó índice) a cobertura (*cobObjSlim*) da cidade de *Cristina*. Dessa forma, o nó representado pela cidade de *Cristina* não precisa ser visitado.

3.4 Considerações Finais

O novo operador de junção chamado *Junção Canalizada*, responde a consultas do tipo "Quais são as cidades que estão até 16 km da rota entre Pouso Alegre / MG e Itajubá / MG" como mostrado nas seções ateriores.

Os elementos e as relaçãoes matemáticas envolvidas nos cálculos usados para determinar os objetos que são respostas fazem com que o operador seja exato nas suas conclusões.

A indexação dos objetos do domínio aliado às podas realizadas fazem com que a abordagem usando a *Slim-Tree* execute menos acessos a disco e menos cálculos de distância em relação à abordagem usando processamento sequencial.

O capítulo seguinte mostra os resultados obtidos com a implementação do operador de *Junção Canalizada* usando o porcessamento sequencial e a abordagem usando a indexação pela *Slim-Tree*.

Capítulo 4 Experimentos

4.1 Considerações Iniciais

Neste capítulo serão apresentados a metodologia e os resultados dos experimentos realizados para a validação do *Operador de Junção Canalizada*.

Em cada medida, o experimento representa a média de 500 execuções do algoritmo de *Junção Canalizada* por abrangência.

Para realizar os experimentos com o algoritmo, são necessários dois conjuntos de dados: pontos do domínio e pontos da rota. Os pontos do domínio são armazenados no método de acesso e os pontos da rota são armazenados em uma lista ordenada.

Nos pontos do domínio utilizou-se duas bases de dados reais. A primeira base de dados contendo o nome, a latitude e a longitude das 5.507 cidades do território brasileiro [30]. Essa base foi usada em tempo de desenvolvimento para avaliar a coerência das respostas retornadas pelo algoritmo de junção canalizada por abrangência usando a *Geometria Euclidiana*. A avaliação foi feita de forma empírica confrontando-se as respostas das junções realizadas com um sistema de mapeamento geográfico.

A segunda base de dados contém 3.976.695 pontos de latitudes e longitudes do planeta Terra extraídos da base da Agência Nacional de Inteligência Geoespacial do governo dos Estados Unidos da América (antiga Agência Nacional de Mapemento e Imagens) [31]. Para realizar comparações esta base usou a *Geometria Esférica* e foi indexada usando o método de acesso métrico *Slim-Tree* detalhado na seção 2.4 e um método de acesso Sequencial. Esses métodos de acesso foram configurados com os tamanhos de blocos: 512 Bytes, 1024 Bytes, 2048 Bytes e 4096 Bytes.

Para os pontos das rotas utilizou-se dados gerados artificialmentre já que não existiam rotas geradas por usuários para a base [31].

A geração de rotas inicia com um ponto chamado sementre e sorteado aleatoriamentre da base [31]. Foram sorteadas 500 sementres que serão os pontos iniciais de cada rota formando as 500 rotas

de cada medida do experimento.

O próximo ponto aleatório na rota deve respeitar uma distância máxima para verificar se essa distância impacta no comportamento do algoritmo. Foram usadas duas configurações de distâncias máximas entre os pontos das rotas: 10 Km e 100 Km.

Por exemplo, nas rotas com distância máxima de 10 Km, o processo de escolha aleatório do próximo ponto realiza uma consulta por abrangência na base [31] usando o ponto anterior e o raio de 10 Km. Essa consulta pode retornar vários pontos como candidatos, sendo que um será escolhido aleatoriamentre como próximo ponto na rota. Apenas por questões de otimização de tempo a consulta por abrangência na base de dados [31] usa o método de acesso métrico *Slim-Tree*.

Um outro fator que poderia promover impacto no comportamento do algoritmo de *Junção Canalizada* por abrangência é a quantidade de pontos na rota. Assim, foram usadas quatro configurações de quantidades de pontos na rota: 10, 100, 1.000 e 2.000.

O algoritmo de *Junção Canalizada* foi implementado usando a biblioteca (*GBDI Arboretum*) [32], composta por vários métodos de acesso métrico implementados, criada pelo *Grupo de Base de Dados e Imagens* (*GBDI*) do Instituto de Ciências Matemáticas e de Computação da USP - São Carlos / SP.

A estrutura da biblioteca é composta por três camadas: a camada de disco, a camada de estrutura e a camada de usuário. A camada de usuário possui os objetos que serão indexados e a métrica que será usada para a indexação. Ela possui duas interfaces definidas que são *stObject* e *stMetricEvaluator* que devem ser implementadas pelo usuário. A camada de estrutura possui a implementação das estruturas de indexação, ela é responsável pelo processamento da indexação, como por exemplo, a *Slim-Tree*. Ela possui duas classes templates *stMetricTree* e *stResult* que são a base da implementação das estruturas de indexação e são o centro do funcionamento da biblioteca. A classe *stMetricTree* é a classe base para todos os *MAMs* implementados enquanto a classe *stResult* é responsável por encapsular o resultado da consulta. Por fim, a camada de disco é responsável pelo armazenamento das páginas (nós) gerados pela camada de estrutura. As classes *stPageManager* e *stPage* provêem os serviços de gerenciamento de páginas no disco para a camada de estrutura. Assim, a classe *stPageManager* abstrai todos os dispositivos de armazenamento como um conjunto de serviços que podem ser usados através da camada de estrutura.

O computador usado nos experimentos contém as seguintes características: processador intel core i7-930 com 4 núcleos físicos e 4 núcleos lógicos (total de 8 núcleos); 32KB de *Cache* L1 por núcleo, 256 KB de *L2* por núcleo físico e 8MB de *L3* compartilhado por todos os núcleos; 4GB de Memória RAM em *single channel*; 500GB de disco rígido em Serial ATA II (3Gb/s). O sistema operacional instalado na computador é o GNU/LINUX distribuição *Kubuntu* versão 11.10 com kernel 3.0 [33].

Cada diretório da tabela 4.1 contém 500 arquivos de rotas, totalizando 16.000 rotas diferentres

contendo em cada uma delas 10, 100, 1.000 ou 2.000 pontos que possuem como distância máxima entre cada um dos pontos 10 ou 100 Km.

Diretório	Arquivos	Pontos	Distância(Km)	Tamanho(KB)
config512_10p_10r	500	10	10	127
config512_10p_100r	500	10	100	127
config512_100p_10r	500	100	10	1.000
config512_100p_100r	500	100	100	1.000
config512_1000p_10r	500	1.000	10	9.500
config512_1000p_100r	500	1.000	100	9.500
config512_2000p_10r	500	2.000	10	19.400
config512_2000p_100r	500	2.000	100	19.400
config1024_10p_10r	500	10	10	127
config1024_10p_100r	500	10	100	127
config1024_100p_10r	500	100	10	1.000
config1024_100p_100r	500	100	100	1.000
config1024_1000p_10r	500	1.000	10	9.500
config1024_1000p_100r	500	1.000	100	9.500
config1024_2000p_10r	500	2.000	10	19.400
config1024_2000p_100r	500	2.000	100	19.400
config2048_10p_10r	500	10	10	127
config2048_10p_100r	500	10	100	127
config2048_100p_10r	500	100	10	1.000
config2048_100p_100r	500	100	100	1.000
config2048_1000p_10r	500	1.000	10	9.500
config2048_1000p_100r	500	1.000	100	9.500
config2048_2000p_10r	500	2.000	10	19.400
config2048_2000p_100r	500	2.000	100	19.400
config4096_10p_10r	500	10	10	127
config4096_10p_100r	500	10	100	127
config4096_100p_10r	500	100	10	1.000
config4096_100p_100r	500	100	100	1.000
config4096_1000p_10r	500	1.000	10	9.500
config4096_1000p_100r	500	1.000	100	9.500
config4096_2000p_10r	500	2.000	10	19.400
config4096_2000p_100r	500	2.000	100	19.400

Tabela 4.1: Configurações dos diretórios com arquivos de rotas.

Método Acesso	Tamanho bloco	Altura	Índices	Folha	Total Blocos	Tamanho (MB)
Slim-Tree	512	8	86.195	483.015	569.210	277,9
Slim-Tree	1.024	6	16.101	216.037	232.138	226,7
Slim-Tree	2.048	5	3.429	102.677	106.106	207,2
Slim-Tree	4.096	4	752	47.845	48.597	189,8
Sequencial	512	-	-	-	233.896	114,2
Sequencial	1.024	-	-	-	110.451	107,9
Sequencial	2.048	-	-	-	55.226	107,9
Sequencial	4.096	-	-	-	27.423	107,1

Tabela 4.2: Configurações dos Métodos de Acessos

A tabela 4.2 apresenta as configurações dos métodos de acesso utilizados nos experimentos. Os métodos de acesso usados foram a *Slim-Tree* e Sequencial. Sobre cada método de acesso são apre-

sentados o tamanho do bloco usado, a altura (somentre *Slim-Tree*), quantidade de blocos índices (*somentre Slim-Tree*), quantidade de blocos folhas (*somentre Slim-Tree*), total de blocos e tamanho do arquivo em MB.

4.2 Resultados

Os valores apresentados nesta seção são os resultados da média do processamento de 500 junções sobre cada uma das 16.000 configurações de rotas distintas.

Inicialmentre a *Junção Canalizada* foi implementada em processamento sequencial para validação do operador. A tabela 4.3 apresenta os valores médios da execução do operador usando o processamento sequencial variando as configurações detalhadas nas colunas. A 1ª coluna apresenta o item que foi aferido, a 2ª coluna indica a distância máxima entre cada ponto em quilômetros (Km) na rota. Em seguinda são apresentados os valores obtidos: para 10 pontos na rota(3ª coluna); para 100 pontos na rota (4ª coluna); para 1.000 pontos na rota (5ª coluna); e para 2.000 pontos na rota (6ª coluna).

Item Aferido	Máx.Dist.Rota	10 pontos	100 pontos	1.000 pontos	2.000 pontos
acesso512	10Km ou 100Km	233.896	233.896	233.896	233.896
acesso1024	10Km ou 100Km	110.451	110.451	110.451	110.451
acesso2048	10Km ou 100Km	55.226	55.226	55.226	55.226
acesso4096	10Km ou 100Km	27.423	27.423	27.423	27.423
cálc.dist.512	10Km ou 100Km	43.738.442	401.598.422	3.980.198.222	7.956.420.222
cálc.dist.1024	10Km ou 100Km	43.738.442	401.598.422	3.980.198.222	7.956.420.222
cálc.dist.2048	10Km ou 100Km	43.738.442	401.598.422	3.980.198.222	7.956.420.222
cálc.dist.4096	10Km ou 100Km	43.738.442	401.598.422	3.980.198.222	7.956.420.222
tempo512	10Km	34.470	367.190	4.117.630	8.429.330
tempo1024	10Km	34.200	368.460	4.121.580	8.526.560
tempo2048	10Km	36.470	368.350	4.102.790	8.536.190
tempo4096	10Km	34.460	366.080	4.114.580	8.594.450
tempo512	100Km	36.690	369.780	4.274.470	8.484.160
tempo1024	100Km	34.240	366.260	4.143.110	8.481.920
tempo2048	100Km	34.340	369.080	4.074.650	8.446.460
tempo4096	100Km	34.010	365.900	4.130.510	8.429.460

Tabela 4.3: Valores aferidos da *Junção Canalizada* usando processamento *Sequencial*.

Analizando a tabela 4.3 pode-se notar que a quantidade de acessos a disco varia em função da quantidade de blocos existentre no arquivo sequencial, já que todos os blocos devem ser visitados. Pode-se ver também que a quantidade de cálculo de distância varia em função da quantidade de pontos na rota, pois corresponde ao produto da quantidade de pontos na rota e a quantidade de pontos no arquivo sequencial. Consequentementre, o tempo está variando em função da quantidade de pontos na rota, já que aumenta na medida em que a quantidade de cálculos de distância também aumenta.

A tabela 4.4 apresenta os valores médios da execução do operador de *Junção Canalizada* usando

o método de acesso métrico *Slim-Tree* variando configurações detalhadas nas colunas. A 1ª coluna apresenta o tamanho do bloco usado na estrutura de indexação, a 2ª coluna apresenta a quantidade de pontos envolvidos na junção, a 3ª coluna indica a distância máxima entre cada ponto em quilômetros (Km) na rota. Em seguida são apresentados os valores quantitativos: na 4ª coluna as médias de acessos a disco; na 5ª coluna as médias de cálculos de distância; e na 6ª coluna as médias de tempo de processamento em milesegundos (ms).

Tamanho bloco	Pontos	Cobertura(Km)	Disco	Cálculos	Tempo (ms)
512	10	10	92.654	581.430	365
512	10	100	98.642	809.352	524
512	100	10	92.714	737.291	536
512	100	100	105.818	5.746.499	4.832
512	1.000	10	95.167	6.625.481	6.805
512	1.000	100	122.10	156.638.855	155.425
512	2.000	10	95.131	19.088.433	20.835
512	2.000	100	130.494	383.140.179	563.570
1024	10	10	30.798	331.200	183
1024	10	100	33.152	540.661	326
1024	100	10	31.104	460.922	314
1024	100	100	35.991	5.130.228	4.252
1024	1.000	10	31.665	5.277.091	5.390
1024	1.000	100	42.914	154.591.515	153.067
1024	2.000	10	32.247	16.841.628	18.908
1024	2.000	100	45.962	402.884.787	422.230
2048	10	10	10.948	226.065	112
2048	10	100	11.995	427.572	253
2048	100	10	11.093	349.767	232
2048	100	100	13.269	4.974.448	4.136
2048	1.000	10	11.440	5.294.479	5.412
2048	1.000	100	16.621	159.814.187	158.550
2048	2.000	10	11.416	15.803.478	17.687
2048	2.000	100	17.811	399.326.462	420.394
4096	10	10	3.661	156.730	68
4096	10	100	4.123	352.254	200
4096	100	10	3.711	278.006	181
4096	100	100	4.714	4.987.177	4.134
4096	1.000	10	3.862	5.038.917	5.155
4096	1.000	100	6.137	152.991.877	151.676
4096	2.000	10	3.947	14.818.748	16.443
4096	2.000	100	6.766	396.256.096	410.485

Tabela 4.4: Valores aferidos da *Junção Canalizada* usando *Slim-Tree*.

A figura 4.1 apresenta graficamentre os valores das tabelas 4.3 e 4.4 para rotas geradas com distância máxima de 10 Km. A primeira coluna: gráficos (a), (c) e (e) são valores aferidos para o operador, implementado usando o método de acesso métrico *Slim-Tree*, e a segunda coluna: gráficos (b), (d) e (f) são valores aferidos para o operador, implementado usando o processamento sequencial.

A primeira linha da figura 4.1 (a) e (b) apresenta os valores de acesso a disco que se mantêm razoavelmentre linear em função do tamanho do bloco. A quantidade de pontos na rota não altera o

comportamento do gráfico. Nota-se que método de acesso métrico *Slim-Tree* sempre visitou menos blocos do que o arquivo sequencial.

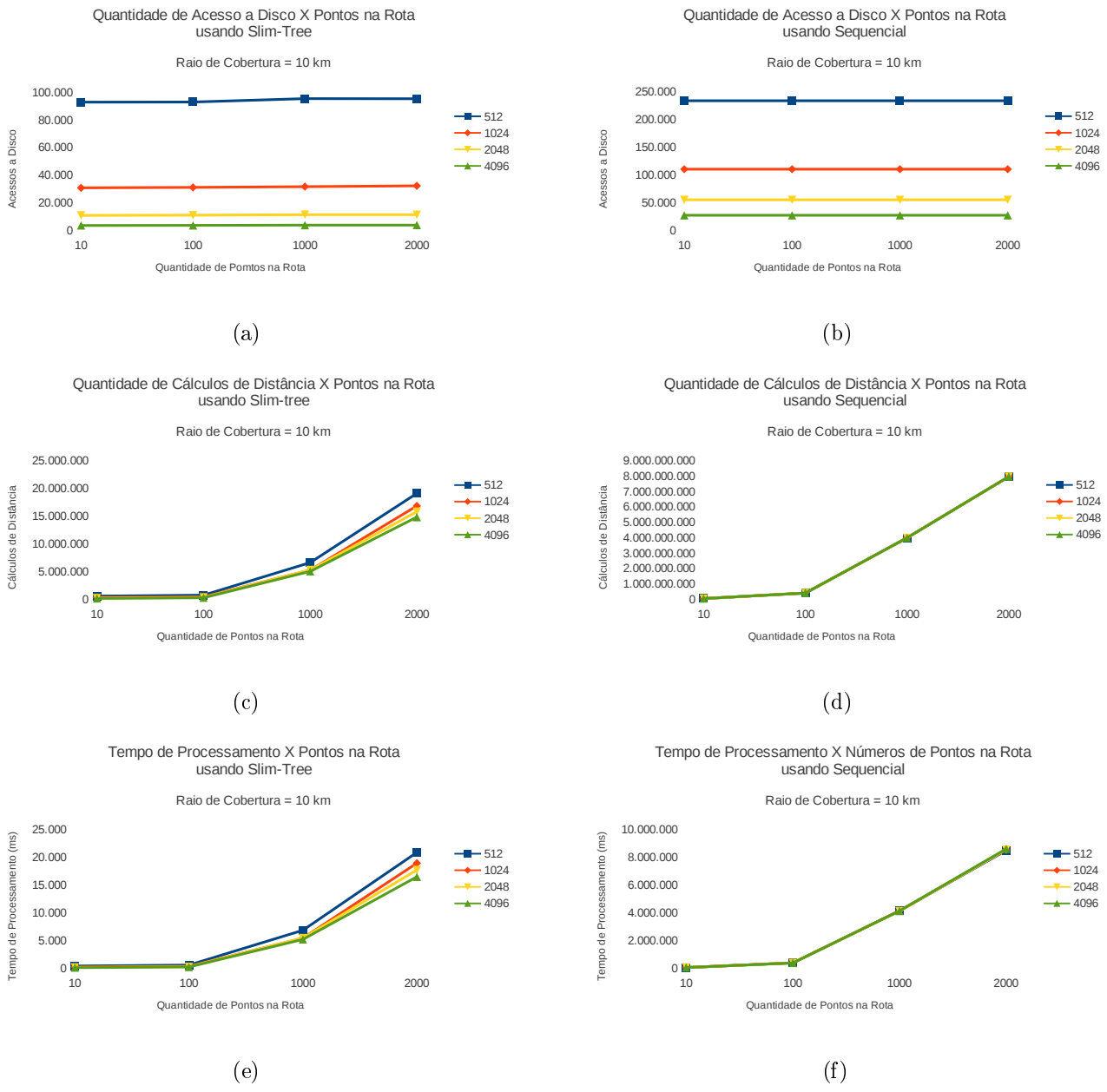

Figura 4.1: Experimentos da *Junção Canalizada* para rotas geradas com distância máxima de 10 Km.

A segunda linha da figura 4.1 (c) e (d) apresenta os valores de cálculos de distância que aumentam em função da quantidade de pontos na rota. O tamanho do bloco provoca distinção de valores no método de acesso métrico *Slim-Tree* quando tem 1.000 e 2.000 pontos na rota. O processamento sequencial sempre executou mais cálculos de distâncias que o método de acesso métrico *Slim-Tree*.

A terceira linha da figura 4.1 (e) e (f) apresenta os valores de tempo, que aumentam em função da quantidade de pontos na rota. O tamanho do bloco provoca distinção de valores no método de

acesso métrico *Slim-Tree* quando tem 1.000 e 2.000 pontos na rota. Pode-se concluir que o tempo de processamento desta junção é influenciado pelo número de cálculos de distância, já que possuem comportamento semelhante.

A figura 4.2 apresenta graficamente os valores das tabelas 4.3 e 4.4 para rotas geradas com distância máxima de 100 Km. A primeira coluna: gráficos (a), (c) e (e) são valores aferidos no método de acesso métrico *Slim-Tree* e a segunda coluna: gráficos (b), (d) e (f) são valores aferidos no processamento sequencial.

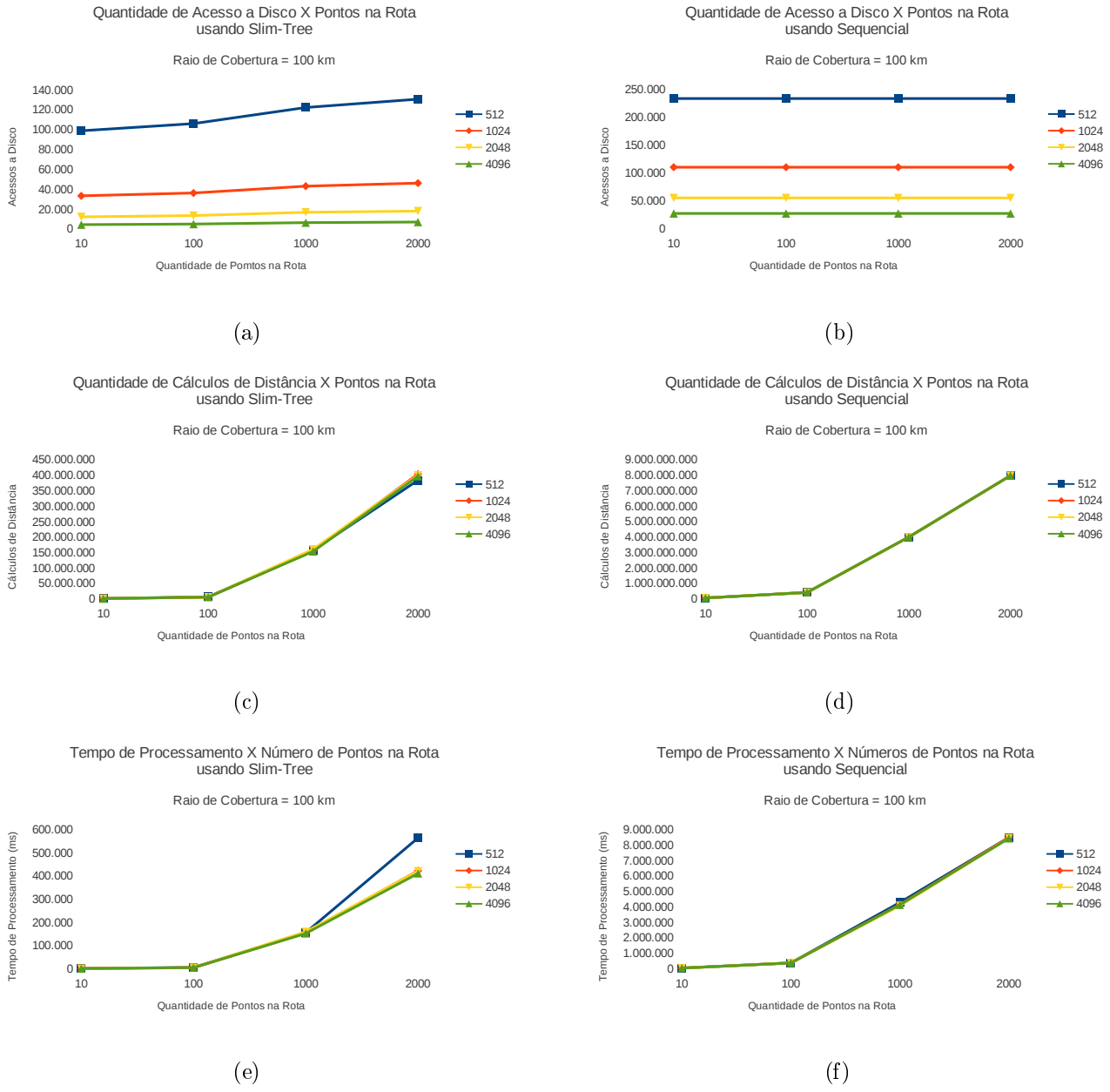

Figura 4.2: Experimentos da *Junção Canalizada* para rotas geradas com distância máxima de 100 Km.

Nas linhas da figura 4.2 pode-se ver os mesmos parâmetros definidos para a figura 4.1. Os gráficos

(a) e (b) apresentam a quantidade de acessos a disco; os gráficos (c) e (d) apresentam a quantidade de cálculos de distância realizados; e os gráficos (e) e (f) apresentam os valores para os tempos obtidos.

Analizando os gráficos da figura 4.1 e da figura 4.2 é possível notar que houve um aumento nos valores representados no eixo *Y*. Dessa forma, pode-se concluir que ocorreu um aumento nos valores aferidos quando a distância máxima entre os pontos das rotas geradas também aumentou.

As tabelas 4.5 e 4.6 apresentam a quantidade de vezes que o tempo de processamento do método de acesso métrico *Slim-Tree* foi melhor que o processamento sequencial nos experimentos realizados com o operador de *Junção Canalizada*. A tabela 4.5 apresenta os resultados dos experimentos com rotas geradas com distância máxima de 10 Km entre os pontos e a tabela 4.6 apresenta os resultados dos experimentos com rotas geradas com distância máxima de 100 Km entre os pontos.

tempo em blocos	**10 pontos**	**100 pontos**	**1000 pontos**	**2000 pontos**
tempo512	94,44	685,06	605,09	404,58
tempo1024	186,89	1.173,44	764,67	450,95
tempo2048	325,63	1.587,72	758,09	482,63
tempo4096	506,76	2.022,54	798,17	522,68

Tabela 4.5: Ganho no tempo usando a *Slim-Tree* sobre o processamento sequencial em rotas com distância máxima de 10 Km.

O menor ganho da *Junção Canalizada* em rotas com distância máxima de 10 Km entre os pontos está na configuração com tamanho de bloco 512 e quantidade de 10 pontos na rota, onde a otimização usando a *Slim-Tree* é 94 vezes mais rápida que o arquivo sequencial. O maior ganho da junção canalizada na mesma configuração de rotas está na configuração com tamanho de bloco 4.096 e quantidade de 100 pontos na rota em que a otimização usando a *Slim-Tree* é 2.022 vezes mais rápida que o arquivo sequencial. Em média, nesses experimentos, a otimização usando a *Slim-Tree* é 563 vezes mais rápida que o arquivo sequencial para o processamento das mesmas junções.

tempo em blocos	**10 pontos**	**100 pontos**	**1.000 pontos**	**2.000 pontos**
tempo512	70,02	76,53	27,50	15,05
tempo1024	105,03	86,14	27,07	20,09
tempo2048	135,73	89,24	25,70	20,09
tempo4096	170,05	88,51	27,23	20,54

Tabela 4.6: Ganho no tempo usando a *Slim-Tree* sobre o processamento sequencial em rotas com distância máxima de 100 Km.

O menor ganho da junção canalizada em rotas com distância máxima de 100 Km entre os pontos está na configuração com tamanho de bloco 512 e quantidade de 2.000 pontos na rota em que a otimização usando a *Slim-Tree* é 15 vezes mais rápida que o arquivo sequencial. O maior ganho da junção canalizada na mesma configuração de rotas está na configuração com tamanho de bloco 4.096

e quantidade de 10 pontos na rota em a que otimização usando a *Slim-Tree* é 170 vezes mais rápida que o arquivo sequencial. Em média, nesses experimentos, a otimização usando a *Slim-Tree* é 48 vezes mais rápida que o arquivo sequencial.

Dessa forma, foram apresentados experimentos que indicam que a *Junção Canalizada* usando a estrutura métrica *Slim-Tree* é a melhor opção de implementação para esse tipo de junção em espaço de dados que usam a *Geometria Esférica*. As podas realizadas nesta implementação, utilizando a desigualdade triangular, fazem com que seus resultados sejam expressivos.

Capítulo 5 Conclusão

5.1 Principais Contribuições

As principais contribuições deste trabalho foram:

1. Novo operador chamado *Junção Canalizada* por abrangência;
2. Validação do operador *Junção Canalizada* por abrangência, através do algoritmo em arquivo sequencial;
3. Otimização do algoritmo do operador de *Junção Canalizada* usando a estrutura métrica *Slim-Tree*;

5.2 Contribuições Secundárias

As contribuições secudárias deste trabalho foram:

1. Aplicação do novo operador de *Junção Canalizada* em domínio de dados sujeitos a *Geometria Euclidiana* e a *Geometria Esférica*;
2. Os experimentos com o operador de *Junção Canalizada* em rotas geradas com distância máxima de 10 km entre os pontos mostraram que a otimização usando *Slim-Tree* é, na média, 563 vezes mais rápida que o arquivo sequencial.
3. Os experimentos com o operador de *Junção Canalizada* em rotas geradas com distância máxima de 100 km entre os pontos mostraram que a otimização usando *Slim-Tree* é, na média, 48 vezes mais rápida que o arquivo sequencial.
4. Proposta de três podas na estrutura *Slim-Tree* para a melhoria do algoritimo do operador de *Junção Canalizada*.

5. Implementação do operador de *Junção Canalizada* através da biblioteca de uso científico chamada *Arboretum*.

5.3 Trabalhos Futuros

Como formas de continuidade deste trabalho cita-se:

1. Criação ou definição de uma nova estrutura para rotas que auxilie o operador de *Junção Canalizada*;

2. Implementação do opreador de *Junção Canalizada* usando o método de acesso espacial *R-Tree*;

3. Avaliação do comportamento do algoritmo do operador de *Junção Canalizada* utilizando rotas geradas por usuários informando um ponto de partida e um ponto de destino;

4. Avaliação do algoritmo do operador de *Junção Canalizada* usando as bases de dados de aparelhos de *Sistemas de Posicionamento Global* (GPS – *Global Positioning System*).

Referências Bibliográficas

[1] Antonin Guttman. Readings in database systems (3rd ed.). chapter R-trees: a dynamic index structure for spatial searching, pages 90–100. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[2] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. *SIGMOD Rec.*, 22(2):237–246, June 1993.

[3] C. Traina, Jr., A. Traina, C. Faloutsos, and B. Seeger. Fast indexing and visualization of metric data sets using slim-trees. *IEEE Trans. on Knowl. and Data Eng.*, 14(2):244–260, March 2002.

[4] Walid G. Aref Yasin N. Silva and Mohamed H. Ali. The similarity join database operator. *ICDE Conference - IEEE*, pages 892–903, 2010.

[5] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, pages 107–141, New York, NY, USA, 1970. ACM.

[6] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publisher, 2000.

[7] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1091–1095, June 2007.

[8] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[9] Priti Mishra and Margaret H. Eich. Join processing in relational databases. *ACM Comput. Surv.*, 24(1):63–113, March 1992.

[10] Caetano Traina, Jr., Agma J. M. Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *Proceedings of the 7th In-*

ternational Conference on Extending Database Technology: Advances in Database Technology, EDBT '00, pages 51–65, London, UK, UK, 2000. Springer-Verlag.

[11] Douglas Comer. Ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, June 1979.

[12] Ramez Elmasri and Shamkant B. Navathe. *Sistemas de Banco de Dados*. Addison Wesley, 2005.

[13] Enzo Seraphim. *Operadores Binários para Consultas de Similaridade em Banco de Dados Multimídia*. PhD thesis, USP - São Carlos, 2005.

[14] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, May 1990.

[15] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 507–518, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[16] Edwin H. Jacox and Hanan Samet. Metric space similarity joins. *ACM Trans. Database Syst.*, 33(2):7:1–7:38, June 2008.

[17] Edwin H. Jacox and Hanan Samet. Spatial join techniques. *ACM Trans. Database Syst.*, 32(1), March 2007.

[18] Ming-Ling Lo and Chinya V. Ravishankar. Spatial hash-joins. *SIGMOD Rec.*, 25(2):247–258, June 1996.

[19] Volker Gaede. Optimal redundancy in spatial database systems. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases*, SSD '95, pages 96–116, London, UK, UK, 1995. Springer-Verlag.

[20] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, June 1998.

[21] Marco Patella Paolo Ciaccia and Pavel Zezula. M-tree: An e cient access method for similarity search in metric spaces. *Proc. Int'l Conf. Very Large Databases (VLDB)*, pages 426–435, 1997.

[22] Christian Böhm, Bernhard Braunmüller, Florian Krebs, and Hans-Peter Kriegel. Epsilon grid order: an algorithm for the similarity join on massive high-dimensional data. *SIGMOD Rec.*, 30(2):379–388, May 2001.

[23] Jens-Peter Dittrich and Bernhard Seeger. Gess: a scalable similarity-join algorithm for mining large data sets in high dimensional spaces. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 47–56, New York, NY, USA, 2001. ACM.

[24] Rodrigo Paredes and Nora Reyes. List of twin clusters: A data structure for similarity joins in metric spaces. In *Proceedings of the First International Workshop on Similarity Search and Applications (sisap 2008)*, SISAP '08, pages 131–138, Washington, DC, USA, 2008. IEEE Computer Society.

[25] Edgar Chávez and Gonzalo Navarro. A compact space decomposition for effective metric indexing. *Pattern Recogn. Lett.*, 26(9):1363–1376, July 2005.

[26] Ke Deng, Kexin Xie, Kevin Zheng, and Xiaofang Zhou. Trajectory indexing and retrieval. In Yu Zheng and Xiaofang Zhou, editors, *Computing with Spatial Trajectories*, pages 35–60. Springer New York, 2011.

[27] Harold Eichholtz Wolfe. *Introduction to Non-Euclidean Geometry*. Mill Press, 2007.

[28] Henry Manning. *Non-Euclidean Geometry*. 2005.

[29] S. Gottwald and W. Gellert. *The VNR concise encyclopedia of mathematics*. Van Nostrand Reinhold, 1989.

[30] Departamento de Informática do SUS. Cadastros nacionais datasus. `http://www2.datasus.gov.br/DATASUS/index.php?area=040206&item=6`, 01 2009.

[31] National Imagery and Mapping Agency's (NIMA). Geonet names server (gns). `http://gcmd.nasa.gov/records/GEONET.html`, 06 2009.

[32] Grupo de Bases de Dados e Imagens. Gbdi - arboretum. `http://www.gbdi.icmc.usp.br/old/arboretum/`, 02 2012.

[33] Canonical Ltd. Kubuntu - friendly computing. `http://www.kubuntu.org`, 02 2012.